国家注册外包专业认证指定系列教材

# 外包软件测试工程师基础教程

信必优技术学院研发部 编著



清华大学出版社

国家注册外包专业认证指定系列教材

# 外包软件测试工程师基础教程

信必优技术学院研发部　编著

清华大学出版社
北　京

## 内 容 简 介

本书是全国网络与信息技术培训项目（NTC）——注册外包专业认证（软件测试工程师初级）的指定教材，全书围绕行业需求和认证考试要求，介绍了作为一名合格的服务外包企业软件测试工程师所必须掌握的理论知识，全面指导软件测试的各个概念、测试流程，以及部分测试文档的写作，参照从业人员的经验，告诉学员如何成为一名合格的服务外包软件测试工程师。

本书是外包软件测试工程师认证考试的必读教材，也可作为大专院校计算机相关专业的参考用书。



**图书在版编目(CIP)数据**

外包软件测试工程师基础教程/信必优技术学院研发部编著.—北京：清华大学出版社，2009.7
（国家注册外包专业认证指定系列教材）
ISBN 978-7-302-20552-4

Ⅰ. 外…　Ⅱ. 信…　Ⅲ. 软件—测试—工程技术人员—资格考核—教材　Ⅳ. TP311.5

中国版本图书馆 CIP 数据核字（2009）第 102458 号

**责任编辑：**周　菁
**责任校对：**王荣静
**责任印制：**
**出版发行：**清华大学出版社　　**地　址：**北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编：**100084
**社 总 机：**010-62770175　　**邮　购：**010-62786544
**投稿与读者服务：**010-62776969，c-service@tup.tsinghua.edu.cn
**质量反馈：**010-62772015，zhiliang@tup.tsinghua.edu.cn
**印 刷 者：**
**装 订 者：**
**经　销：**全国新华书店
**开　本：**185×230　**印　张：**13.5　**防伪页：**1　**字　数：**288 千字
**版　次：**2009 年 6 月第 1 版　**印　次：**2009 年 6 月第 1 次印刷
**印　数：**
**定　价：**　元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103　产品编号：

# 序　　言

根据“国务院大力发展职业教育的决定”（国发［2005]35 号），中共中央、国务院《关于进一步加强人才工作的决定》，人事部关于国家对专业技术人员加强培训且须持证上岗等文件精神，信息产业部根据国家职业技术标准要求[①]，推出了全国网络与信息技术培训项目（NTC，www.ntc.org.cn），旨在培养国家信息化专业技术人才及管理人才，树立 IT 行业的国家标准，考核通过后颁发信息产业部技术资质证书，可作为专业技术水平的凭证及从事相关岗位的任职依据。

为了加大培养国内服务外包人才的力度，推进加速服务外包行业发展的理念贯彻全国，全国网络与信息技术培训项目管理中心（NTC-MC）将“注册外包专业认证”纳入现有培训认证项目之列，并审核、颁发信息产业部“全国网络与信息技术培训（NTC）——注册外包专业认证”。专项技术认证证书样本如下图所示。




证书样本图

NTC-MC 采取授权的方式，成立全国注册外包专业认证行业管理中心，负责提供教材、认证推广和课程培训等相关业务，美国信必优技术学院（Symbio Technology Institute，简称 STI，www.outsourcing.org.cn）是 NTC-MC 授权的全国唯一一家“注册外包专业认

---

[①] 本书所提到的人事部、信息产业部经 2008 年十一届全国人民代表大会更名为人力资源和社会保障部、工业和信息化部。由于本书涉及的内容属延用，故用旧名称。

证”行业管理中心。

目前，注册外包专业认证涵盖了信息技术外包（ITO）和业务流程外包（BPO）的各个领域，现已包括七大类专业方向。包括了软件测试工程师、Java 软件开发工程师、.Net 软件开发工程师、嵌入式软件开发工程师、企业网络与系统工程师、呼叫中心技能及国际外包项目管理等专业。

该证书拥有如下特征：

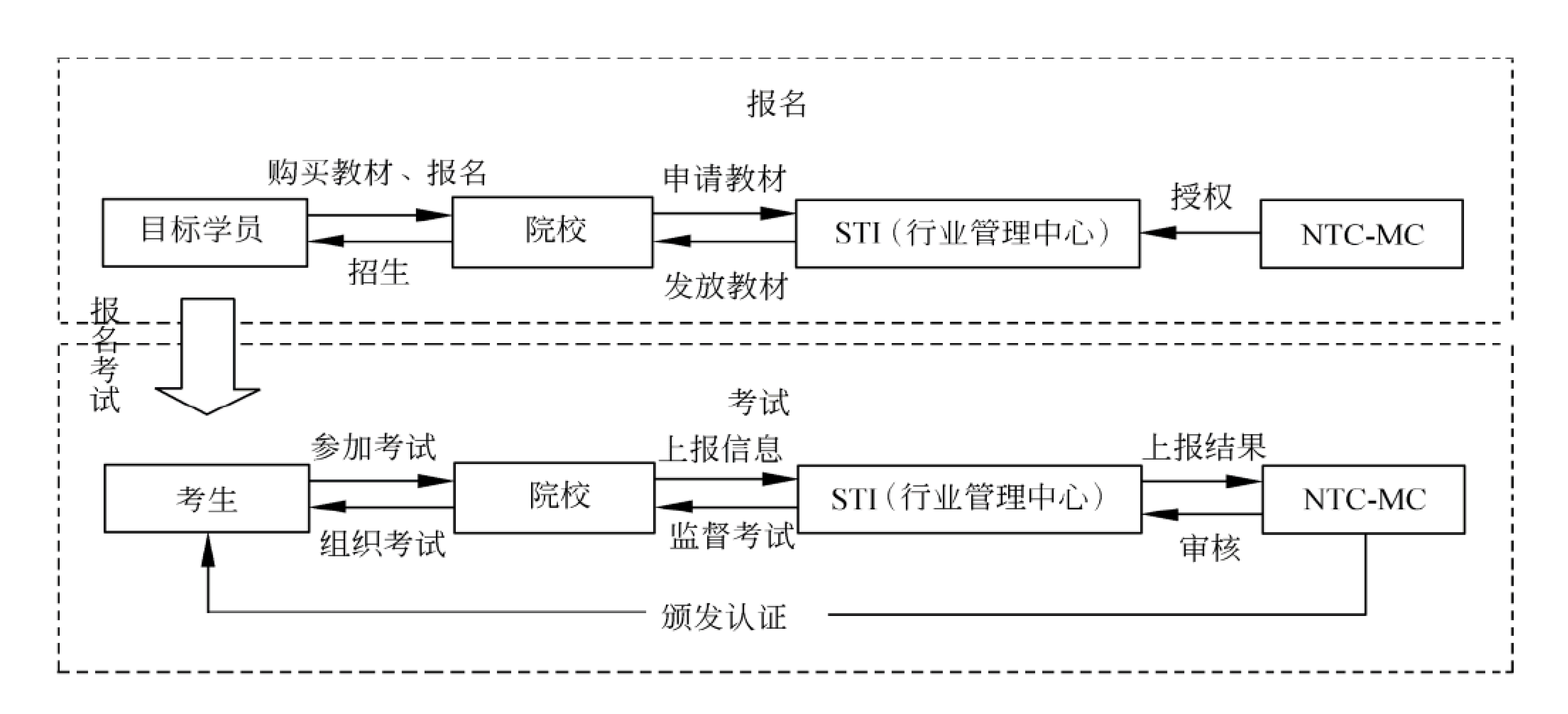
- 国家级——迄今为止，信产部有关部门唯一获批的全国性外包类认证。
- 专业性——迄今为止，国内服务外包领域唯一的专业性资质认证。
- 国际化——迄今为止，外包领域最权威的国际化外包专业认证体系。

根据授课对象，认证考试方式不同，注册外包专业认证的每类证书分为 3 个等级（初级、中级、高级），共计 21 个证书。

**初级证书（Entry Level）**

该级别证书面向的是面临就业压力、职业困扰、职业生涯规划问题的在校大学生。

报名考试方式如下图所示。




报名考试流程示意图

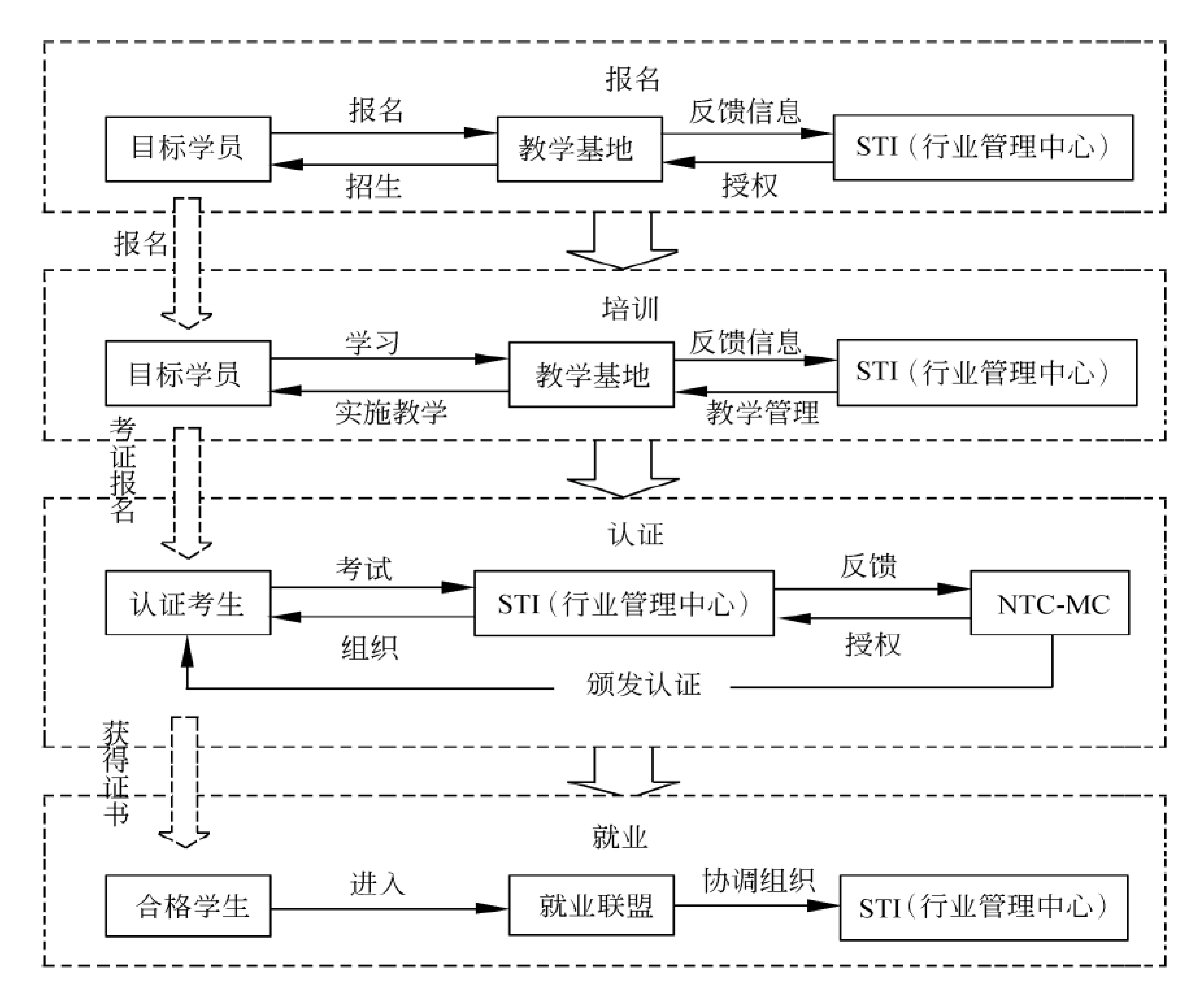
若希望报名参加“注册外包专业认证——初级软件测试工程师”认证考试的学生，可以通过报名和购买教材的方式，获得参加公开课及报名参加初级认证考试的资格。STI 行业管理中心负责协调和组织认证考试，并进行考试监督。信息产业部相关部门审核考试结果后颁发认证，考生可登录到 NTC 官方网站查询考试结果。

**中级证书（Middle Level）**

该级别的证书面向的是大学应届毕业生和具备一定工作经验的社会在职或者求职

人员，解决学生进入服务外包企业的专业技能不足的问题。

认证培训方式如下图所示。




认证培训流程示意图

若希望报名参加“注册外包专业认证——初级软件测试工程师”认证考试的学生，可以通过报名和购买教材的方式，经过一定课时的专业培训，报名参加中级认证考试，考试合格者颁发中级证书并进入就业体系。STI 行业管理中心负责协调和组织认证考试，并进行考试监督。信息产业部相关部门审核考试结果后颁发认证，考生可登录到 NTC 官方网站查询考试结果。

**高级证书（High Level）**

该级别证书面向的是企业在职的高端服务外包人才，解决国际化服务外包市场的人才缺乏问题。通过一定课时的高端培训，并通过高级认证考试，合格者颁发高级证书，并提供厂商认证。

配合国际大厂的厂商认证培训，为国际服务外包公司直接派遣人才，建设国际服务外包人才的高端培训品牌。通过培训的学员，派遣至合作的国际服务外包公司。

认证报名联系方式：
中心：注册外包专业认证行业管理中心
地址：北京市海淀区上地五街 5 号高立二千大厦 1 层（100085）
网址：www.outsourcing.org.cn
电话：010-62968496

注册外包专业认证行业管理中心

# 前　言

在整个软件领域，软件服务外包行业异军突起，从业人员需求大幅度增加。国内大多数城市的软件服务外包业都出现了不同程度的“人才荒”。2008 年北京软件行业人才缺口达 5 万人，上海软件人才缺口是 10 万人，中国市场每年至少存在 50 万软件人才的巨大缺口，而且这个缺口还在以每年 20%的速度递增。“订单充裕，人才缺乏。”众多软件服务外包企业表示，虽然目前很多大学生找不到就业岗位，但企业却招不到合适的人；现今的从业人员大多也不能满足软件服务外包行业的专业要求。

“注册外包专业认证体系”中软件服务外包部分，为学员提供从事软件服务外包必备知识和技能的专业培训，合格者会获得信产部“全国网络与信息技术培训（NTC）——注册外包专业认证”专项技术资质证书，为企业提供对口的服务外包人才，并为企业和专业外包人才搭建一个互信的桥梁，及时解决软件服务外包业的“人才荒”。

软件测试工程师认证项目在“注册外包专业认证”体系中占据极其重要的地位。该项目为在校大学生、应届生及相关求职人员提供软件服务外包行业所需的软件测试知识和职业素质培训、认证及就业服务等，帮助他们在理解和掌握外包领域软件测试专业知识的基础上，加强对软件服务外包企业的工作流程、项目管理方法的认识。最终目的是为软件服务外包企业提供技能和素质兼备的优秀软件测试工程师。

“全国网络与信息技术培训（NTC）——注册外包专业认证”（软件测试工程师）考试科目初级和中级的考试科目，如下表所示。

**注册外包专业认证（软件测试工程师）考试科目**

| 认　证 | 考 试 科 目 |
|---|---|
| 注册外包专业认证——初级软件测试工程师 | 1. 外包软件测试工程师基础教程<br>2. 搭建 Windows 测试环境技术<br>3. Java 面向对象编程基础 |
| 注册外包专业认证——软件测试工程师 | 1. 搭建 Linux 测试环境技术<br>2. 软件测试技术详解及应用<br>3. 软件自动化测试工具实用技术<br>4. 软件测试与质量保证技术 |

《外包软件测试工程师基础教程》一书，是“全国网络与信息技术培训

（NTC）——注册外包专业认证”（初级软件测试工程师）课程体系中的相对重要的基础课程，主要介绍了软件测试最基础的理论知识和外包方面的知识。21 世纪是经济全球化的发展时代，各国和各地区间的产业关联度不断增加，软件产业作为信息产业国际化的排头兵，其国际化设计和本地化处理已经构成了生产过程的两翼，我国的软件产业也不可避免地融入到全球软件产业变革中，一方面，更多的大型跨国软件公司加大对中国市场的投入，对产品和服务本地化的需求快速增加；另一方面，越来越多的国内大型软件公司正加速国际化发展步伐，逐步走出国门，加入全球竞争的行列。软件外包生产作为此种变革的重要衍生，已经成为中国软件产业乃至全球软件产业的生产新模式。

在种类繁多的软件外包服务领域中，外包软件测试作为一个重要的分支，为我国软件企业带来了新的商机。软件外包测试包含较为广泛的内容，例如软件的核心功能测试，软件国际化测试和软件的本地化测试。另外，软件外包测试的实现方式也多种多样，例如外包现场测试（On-site Outsourcing Testing），离岸外包测试（Off-shore Outsourcing Test）等。不同的外包测试内容和实现方式需要选择不同的外包测试服务公司。

通过对软件外包测试内容和实现方式的了解，以及对我国软件外包企业的规模现状分析，中国政府认识到大力发展软件外包测试，是成为进军国际软件外包市场的最佳入口。在此种大形势下，软件外包测试人才供需矛盾突出，人才培养迫在眉睫。美国信必优技术学院通过中国信息产业部 NTC 项目独家授权推出的“注册外包专业认证”教材和认证，为在校大学生、社会求职人士，与外包软件企业搭建了一座互信的桥梁。

本书为注册外包专业认证（软件测试工程师初级）的重要组成部分。全书分为 11 章，分别介绍了“注册外包专业认证”的由来，外包行业的基础知识，以及作为外包软件测试工程师所需要掌握的基本技术技能，职业素质需求。参照业界从业人员的经验，告诉考生如何成为一名合格的软件外包测试工程师。

通过本书的学习，考生除了能多了解一些外包测试以外，还能获得职业发展、快乐工作有关的信息和经验。在未来的职业发展和职业选择中可以少走些弯路，少一些迷茫，少一些浮躁，看到自身的价值所在。

信必优技术学院研发部
2009 年 5 月

# 目　　录

# 第 1 章　软件外包与软件测试行业简介

**学习目标：**

1．了解什么是服务外包

2．了解什么是软件外包

3．理解什么是外包软件测试

## 1.1　服务外包

### 1.1.1　什么是服务外包

服务外包是指企业将其非核心的业务外包出去，利用外部最优秀的专业化团队来承接其业务，从而使其专注核心业务，达到降低成本、提高效率、增强企业核心竞争力和对环境应变能力的一种管理模式。它包括信息技术外包（ITO）、商业流程外包（BPO）和知识流程外包（KPO）。

服务外包企业系指根据其与服务外包发包商签订的中长期服务合同，向客户提供服务外包业务的服务外包提供商。

服务外包业务系指服务外包企业向客户提供的信息技术外包（ITO）服务和业务流程外包（BPO）服务。包括业务改造外包、业务流程和业务流程服务外包、应用管理和应用服务等商业应用程序外包、基础技术外包（IT、软件开发设计、技术研发、基础技术平台整合和管理整合）等。

### 1.1.2　服务外包内容和分类

**1．服务外包的内容**

（1）信息技术外包（ITO）。

基础技术服务：承接技术研发、软件开发设计、基础技术或基础管理平台整合或管理整合等。

系统操作服务：银行数据、信用卡数据、各类保险数据、保险理赔数据、医疗/体检数据、税务数据、法律数据（包括信息）的处理及整合。

系统应用服务：信息工程及流程设计、管理信息系统服务、远程维护等。

（2）业务流程外包（BPO）。

企业内部管理服务：为客户企业提供企业各类内部管理服务，包括后勤服务、人力

资源服务、工资福利服务、会计服务、财务中心、数据中心及其他内部管理服务等。

企业业务运作服务：为客户企业提供技术研发服务、销售及批发服务、产品售后服务（售后电话指导、维修服务）及其他业务流程环节的服务等。

供应链管理服务：为客户企业提供采购、运输、仓库/库存整体方案服务等。

**2．服务外包的分类**

根据服务外包承接商的地理分布状况，服务外包分为 3 种类型，即离岸外包、近岸外包和境内外包。离岸外包是指转移方与为其提供服务的承接方来自不同国家，外包工作跨境完成；近岸外包是指转移方和承接方来自于邻近国家，近岸国家很可能会讲同样的语言，在文化方面比较类似，并且通常提供了某程度的成本优势；境内外包指转移方与为其提供服务的承接方来自同一个国家，外包工作在境内完成。

### 1.1.3 国家对服务外包的扶持政策

商务部下发了关于做好服务外包“千百十工程”人才培训有关工作的通知。

为促进我国服务外包产业健康快速发展，加强行业的能力建设，加大对服务外包人才培养的支持力度，根据《商务领域人才培训专项资金使用管理暂行办法》（商财发[2006]281 号，以下简称“暂行办法”）以及商务部服务外包“千百十工程”工作的要求，现就规范和加强服务外包“千百十工程”人才培训有关工作通知如下：

一、本通知所称服务外包“千百十工程”人才培训资金（以下简称“服务外包人才培训资金”）系指为实施服务外包“千百十工程”人才培训计划，在商务领域人才培训资金中专项安排的服务外包公共培训资金。

二、使用服务外包人才培训资金的地方政府在已有的财政专项资金中安排相同金额的资金，作为服务外包人才培训配套资金。

三、服务外包人才培训资金的使用必须遵守国家的有关法律、法规和财务制度，严格按照《暂行办法》的要求，坚持科学立项、专家评审、择优支持、公正透明、非营利性的原则。

四、服务外包人才培训资金主要用于促进我国服务外包产业发展，解决服务外包企业人才短缺，支持大学生（含大专，下同）增加服务外包专业知识和技能，鼓励大学生在服务外包企业就业的各类人才培训项目。重点培训对象为大学应届毕业生和尚未就业的大学毕业生，以及服务外包企业新入职员工。

五、服务外包人才培训重点项目包括：根据服务外包企业承接服务外包业务的需求或服务外包发包商提出的承接其发包业务的需求进行的人才定制培训；跨国公司服务外包业务从业人才资质培训；服务外包企业国际认证知识培训、服务外包企业国际认证人才培训、发展服务外包产业急需的储备人才培训；服务外包企业大学生实习项目及勤工俭学培训；服务外包相关法律、行业标准及相关知识产权培训；服务外包企业新入职人员岗前业务技能培训等。

六、对大学应届毕业生和尚未就业的大学毕业生服务外包从业技能培训的资金支持，不超过培训费用的 85%；对服务外包企业新入职员工岗前业务技能培训的资金支持，不超过培训费用的 50%。属于人才定制培训的，学员培训合格并被服务外包企业录用，由服务外包企业返还给学员由其个人承担的培训费用。

七、各地商务主管部门应严格按照《暂行办法》第六条规定的程序，在对企业的申请或当地服务外包产业发展需求进行调研、论证的基础上，确定具体培训项目，形成年度培训计划。年度培训计划应包括如下内容：培训岗位、培训对象、培训内容、培训方式、培训周期、师资来源、培训人数、费用预算等。

八、各地商务部门应本着竞争、择优的原则，选择具体培训项目承办单位，负责培训项目的具体执行。承办单位应具备如下条件：

（一）具有人才培训的从业资格；

（二）具有符合条件的场地、设施和师资力量；

（三）有为承接服务外包企业提供培训的经验；

（四）具有健全的财务制度和合格的财务管理人员。

九、各地商务主管部门应与项目承办单位签订委托培训协议，并就每个培训项目签订合同，规定培训项目的具体要求，明确双方的责任、义务，实行合同制管理。各地还可以制定奖励制度，对公信度好、就业率高的项目承办单位给予奖励。

十、各地商务主管部门应在各地方财政主管部门同意后，于每年三月底之前将本年度服务外包人才培训计划和具体项目建议书报商务部、财政部。经商财政部同意后，商务部、财政部共同下发服务外包人才培训项目计划通知，并由财政部将培训项目所需资金拨付至省级财政部门。2006 年度第四季度的培训计划于十月十日以前上报商务部。

十一、各地商务主管部门应根据《商务领域人才培训专项资金使用管理暂行办法》和本通知的要求，制定资金管理和使用办法实施细则，并报商务部（外资司）备案。细则的主要内容应包括：培训机构的选择标准、培训项目的基本条件、培训费用返还的具体操作办法、资金申报和拨付的时间、程序和方法、对培训机构的监督检查办法等。

十二、本通知自发布之日起实施。

商务部

2006 年 9 月 22 日

## 1.2　软件外包

### 1.2.1　什么是软件外包

软件外包以前就是一些发达国家的软件公司将他们的一些非核心的软件项目通过外包的形式交给人力资源成本相对较低的国家的公司开发以达到降低软件开发成本的目

的。现在不单是发达国家外包给人力资源相对较低的国家来完成软件需求活动，而且还是企业为专注核心竞争力业务和降低软件项目成本将软件项目中的全部或部分工作发包给提供外包服务的企业完成的软件需求活动。其中软件外包包括软件开发、软件测试、本地化测试等方面的外包。

### 1.2.2 中国软件外包的背景

作为全球软件外包行业的新兴力量，中国在这几年里取得了长足的进步。中国市场已经成为具有较大潜力的全球外包市场，而且已经成长起一批颇具规模的中国外包企业。尽管我国软件外包业务启动时间较晚，但一直呈高速增长态势，被公认为是新兴的国际软件外包中心。在对日软件外包市场方面取得了重大成功，对欧美的软件外包市场实现了局部突破，中国正成为迅速崛起的外包新军。

无论是国外看中国，还是国内横向比较，我国软件外包服务都是“利好”市场。从全球范围来看，2007 年，全球软件外包市场规模达到了 328 亿美元，增长率高达 25%，而且这一市场规模还在呈膨胀式增长。到 2011 年，全球软件外包市场规模将达到 800 亿美元，年复合增长率为 19.5%。

据赛迪顾问 2008 年 2 月提供的数据，从 2005 年到 2007 年，我国软件外包市场规模分别是 3.25 亿美元、4.7 亿美元和 6.33 亿美元，年均以 40%的幅度增长，而到 2011 年预计达到 45.6 亿美元。

从软件外包服务的客户市场分布看，我国是日本的最大软件外包服务国，2007 年达到 4.02 亿美元，对日外包一枝独秀。对欧美软件外包市场的份额取得了较快发展，其中美国市场达到 0.87 亿美元，而且呈现快速发展的态势。

较低的人力成本，政府和行业协会的支持，完善的基础设施，巨大的市场发展潜力，使得全世界外包客户纷纷将目光投向中国。经过近几年的不懈努力，国内已经出现了一批软件外包服务的专业公司。2007 年外包收入超过 1000 万美元的软件企业达到 8 家，排名第一的中国外包公司外包年收入达到 3300 万美元。

瑞士日内瓦霍拉西斯战略咨询公司和美国全球投资风险研究所，最近联合推出的《2008 年全球业务外包研究报告》预测中国将在未来 10 年内取代印度，成为全球科技业务外包的最大承接国。中国外包服务行业方兴未艾，前景诱人。

### 1.2.3 发展软件外包的三个阶段

挖掘客户的需求，提升外包服务的层次，增强竞争力，寻求新的附加值，实现从服务提供商到业务合作伙伴的角色转变，将成为现在和今后发展软件外包服务的新动力。

尽管软件外包在制造和建筑等行业已经比较成熟，但是在软件行业却还是新鲜事物。从软件外包服务的发展阶段看，将经过初级孕育、中期扩展和高级完善的“三级跳”发展阶段。

第一阶段是软件外包的初级孕育阶段，软件开发公司仅将少量工作外包，软件外包

服务公司与软件开发公司只是浅层次的从属关系。全球范围内的软件外包大概可以追溯到 20 世纪 80 年代中后期，从国际大型软件公司的软件本地化外包开始的。那时外包服务提供商承担的只是软件程序和文档的本地化翻译工作。这一阶段已经成了历史，它成就了爱尔兰软件产业的后来居上，也催生了国内早期软件本地化服务公司的不断壮大。

第二阶段为软件外包的中期扩展，软件开发公司增大外包工作的种类，一些表现优异的软件外包服务公司成为软件开发公司的优先外包服务公司。这一阶段软件开发公司将软件编码、软件测试、软件本地化交付给外包服务公司完成，外包的主要目的是降低成本和提高质量。软件外包服务公司可以获得更多的外包业务，取得更多的经济收益。但是由于他们不具有任何知识产权，仅靠提供技术和项目管理获得利润，外包服务的生产活动受制于客户的计划，具有较明显的阶段波动性。尽管软件开发公司比以前更重视他们的价值，但是彼此之间仍然是业务合作关系，彼此缺乏充分的信任和依存。

第三阶段是高级完善发展，软件开发公司与外包服务公司形成利益共享、风险共担的业务合作伙伴。外包服务公司不仅为软件开发公司提供全球化发展咨询，还与软件开发公司共同推出创新的行业解决方案，而这项方案是任何一方都不可能自行发展的。这时软件外包的目的是增强核心竞争力，为最终用户提高服务水平。软件开发公司与外包服务公司之间拥有充分的信任和默契，共同促进企业的成功。为达此目标，外包服务公司必须与软件开发公司共同承担责任，深入了解软件开发公司的外包策略、商业文化和企业文化，协助软件开发公司整合其内部资源。外包服务公司除了获得解决方案的服务费用，还拥有软件产品的部分知识产权。

商业应用软件外包现在处于哪个发展阶段呢?在欧美等软件高端市场，软件开发公司经过了 20 世纪 90 年代初的软件本地化外包的成功实践，已经度过了软件外包的第一阶段。进入 21 世纪后，随着技术的快速发展，软件行业的分工越来越细，基本上形成了操作系统、数据库、中间件和应用系统这样一个完整的产业链雏形。但是由于软件应用范围的不断扩大，使得软件设计规模不断膨胀，技术复杂性迅速提升，而且市场竞争不断加剧，软件开发周期逐渐缩短，仅靠软件开发公司的单方面努力已经不能跟上软件变革的步伐，由此催生了软件外包的第二次高潮。当前风靡世界的软件外包正在推动软件外包进入中期扩展的第二个阶段，推动全球软件行业的重新布局。

## 1.3　外包软件测试

### 1.3.1　软件测试的由来

软件测试是伴随着软件的产生而产生的。早期的软件开发过程中，测试的含义比较狭窄，将测试等同于“调试”，目的是纠正软件中已经知道的故障，常常由开发人员自己完成这部分的工作。对测试的投入极少，测试介入也晚，常常是等到形成代码，产品

已经基本完成时才进行测试。

直到 1957 年，软件测试才开始与调试区别开来，作为一种发现软件缺陷的活动。由于一直存在着“为了让我们看到产品在工作，就得将测试工作往后推一点”的思想，测试仍然是后于开发的活动。潜意识里，我们的目的是使自己确信产品能工作。

到了 20 世纪 70 年代，尽管对“软件工程”的真正含义还缺乏共识，但这一词条已经频繁出现。1972 年，在美国北卡罗来纳大学举行了首届软件测试正式会议。1979 年，G.J.Myers 在《软件测试的艺术》（*The Art of Software Testing*）中作出了当时最好的软件测试定义：“测试是为发现错误而执行的一个程序或者系统的过程。”

直到 20 世纪 80 年代早期，“质量”的号角才开始吹响。软件测试定义发生了改变，测试不单纯是一个发现错误的过程，而且包含软件质量评价的内容。软件开发人员和测试人员开始坐在一起探讨软件工程和测试问题。制定了各类标准，包括 IEEE（Institute of Electrical and Electronic Engineers）标准、美国 ANSI（American National Standard Institute）标准以及 ISO（International Standard Organization）国际标准。1983 年，Bill Hetzel 在《软件测试完全指南》（*Complete Guide of Software Testing*）一书中指出：“测试是以评价一个程序或者系统属性为目标的任何一种活动。测试是对软件质量的度量。”Myers 和 Hetzel 的定义至今仍被引用。

20 世纪 90 年代，测试工具终于盛行起来。人们普遍意识到，工具不仅仅是有用的，而且要对今天的软件系统进行充分的测试，工具是必不可少的。

到了 2002 年，Rick 和 Stefan 在《系统的软件测试》（*Systematic Software Testing*）中对软件测试做了进一步定义：“测试是为了度量和提高被测软件的质量，对测试件进行工程设计、实施和维护的整个生命周期过程。”这些经典论著对软件测试研究的理论化和体系化产生了巨大的影响。

近 20 年来，随着计算机和软件技术的飞速发展，软件测试技术研究也取得了很大的突破。测试专家总结了很好的测试模型，比如著名的瀑布模型、V 模型等，在测试过程改进方面提出了 TMM（Testing Maturity Model）的概念，在单元测试、自动化测试、负载压力测试以及测试管理等方面涌现了大量优秀的软件测试工具。

虽然软件测试技术的发展很快，但是其发展速度仍落后于软件开发技术的发展速度，使得软件测试在今天面临着很大的挑战，主要体现在以下几个方面。

- 软件在国防现代化、社会信息化和国民经济信息化的作用越来越重要，由此产生的测试任务越来越繁重。
- 软件规模越来越大，功能越来越复杂，如何进行充分而有效的测试成为难题。
- 面向对象的开发技术越来越普及，但是面向对象的测试技术却刚刚起步。
- 对于分布式系统整体性能还不能进行很好的测试。
- 对于实时系统来说，缺乏有效的测试手段。
- 随着安全问题的日益突出，如何对信息系统的安全性进行有效的测试与评估，成为世界性的难题。

### 1.3.2　软件测试行业发展与现状

在国内，软件测试尚处于起步阶段，但前景是光明的，有越来越多的人开始关注这个行业，因为有越来越多的人已投身到这个行业，也有越来越多的人喜欢这个行业。

**1．国内测试行业现状**

关于国内软件测试行业的现状，国内知名的人才服务机构智联招聘发布的《2009 年度软件测试行业专项调查报告》中，有几个值得注意的数据。

（1）对软件测试重要性的调查结果如图 1-1 所示。68.2%的受访企业认为软件测试非常重要，必须设立专门的测试部门，并将其视为与开发环节同等重要的地位。另外，31.8%的企业选择了比较重要，而认为软件测试只起到“一定作用”或“可有可无”的比例为 0。可见软件测试得到了大部分人的重视。

（2）测试人员所占的比例，如图 1-2 所示。调查数据显示，被调查企业中测试人员与开发人员比例为 1∶5 的企业高达 36.4%，比例为 1∶2 的企业占 31.8%，比例为 1∶1 及以上的企业仅占 31.7%。由此可见，部分公司的测试人员比例仍然偏低。

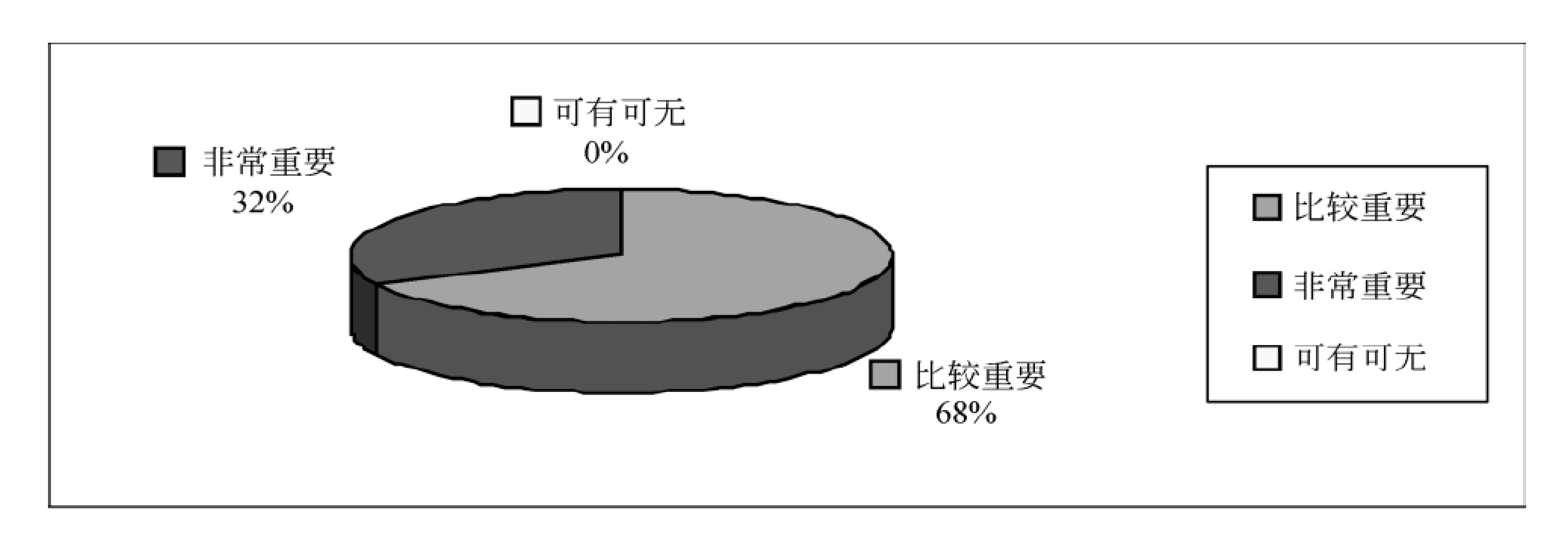



图 1-1　软件测试重要性调查结果

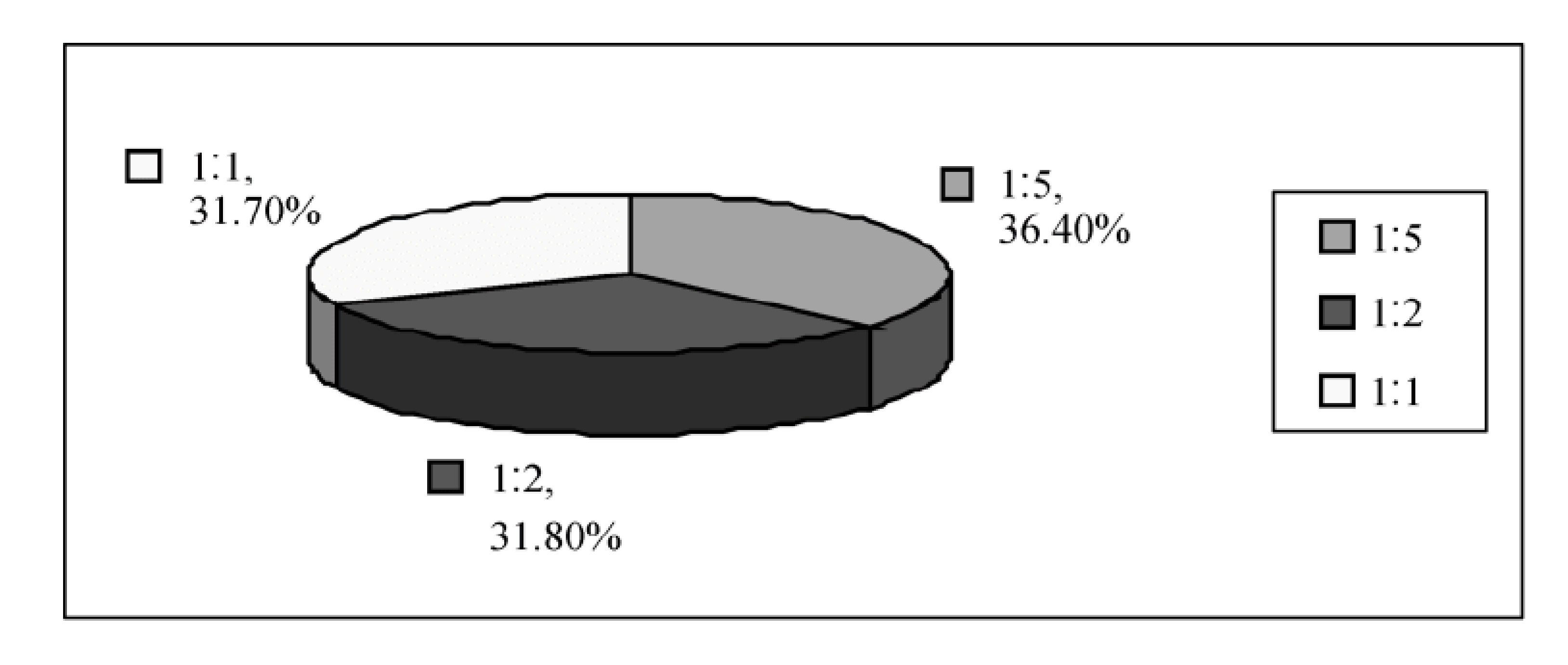



图 1-2　测试人员所占的比例

（3）测试行业的受欢迎程度。数据显示，在面向社会人群的调查中，有87%的被调查者表示出对软件测试行业的青睐。

（4）测试人员的能力情况。调查结果显示，企业在招聘人才时遇到“很多计算机专业应届毕业生缺乏实际经验和动手能力”和“以往做过测试的应聘者并未系统地掌握软件测试流程”问题的比例分别占到了72.7%和59.1%。

从这份报告中的数据大致可以看出，软件质量和软件测试受到越来越多人的重视，测试人员的需求量在增大。与此同时，软件测试人员的能力严重不足。

**2．测试人员的现状**

一方面，由于测试职位比较紧缺，受到大家的重视，测试人员的待遇也开始提高，因此，有不少的软件测试人员开始选择跳槽，不能安下心来努力积累经验、提高自己的能力。

另一方面，软件测试职位受到很多毕业生的青睐，也有很多希望转行到软件测试的人员。这给一些培训机构创造了很好的商机，承诺参加培训课程就能保证就业。这加剧了测试行业的浮躁氛围，很多人仅针对着就业而参加培训，真正静下心来学习软件测试的人不多。

分析近几年来很多测试人员的应聘表现，可以看出其能力不足和浮躁的主要原因有以下几种。

- 基础知识不够扎实：仅仅浮浅地了解一些基本的测试设计方法，并没有深入理解这些基本概念。
- 专业技术不够精通：个人简历上写着精通某某技术或某某工具，但是基本上没有真正地实实在在地应用过。
- 没有建立相对完整的测试体系概念，忽视理论知识：大部分人对软件测试的基本定义和目的不清晰，对自己的工作职责理解不到位。测试理论知识缺乏，认为理论知识没用而没有深入理解测试的基本道理。

这是软件测试行业在中国必然经历的一个不成熟阶段。软件测试行业最终会趋于平静，进入平稳的发展阶段。

**3．软件测试的前景**

Harry Robinson在2007年的时候就曾经对软件测试的未来趋势进行过预测，他认为测试领域在将来会有如下的一些变化。

- 需求工程师、开发人员会成为软件测试人员中的一分子，他们与测试人员之间开始互相帮助。
- 测试的方法日趋完善，Bug预防和早期检查将成为测试工具的主流；通过仿真工具来模拟真实环境进行测试。
- 测试用例的更新变得容易。

- 对测试质量的衡量开始从计算 Bug 数量、测试用例数量转到需求覆盖、代码覆盖等方面。
- 机器将替代测试人员做大部分的工作，更少更精的测试人员开始把注意力集中在更严重的问题上。
- 测试人员将运行更多更好的测试代码。
- 测试执行与测试开发的界限开始模糊。
- 测试与开发的界限开始模糊。
- 顾客反馈与测试合为一体。
- 新的挑战，例如安全测试等新问题开始出现。
- 测试人员获得尊重、测试变得流行。
- 开发人员追求进度，项目开发到最后一刻才加入测试的行为仍然会存在。

在近几年中，可以看到，上面的一些变化已经开始，或者正在开始。例如，人们逐步意识到测试只是事后的检测，这并不能达到消灭缺陷的作用，人们开始转向缺陷产生的早期，开始对缺陷进行预防性的工作。

Harry Robinson 还提出了以下几点迎接测试未来挑战的建议。

- 积极地不满于现状：不要被动接受和满足测试的现状，不要埋头苦“测”，要思考一下“我们在做哪些毫无意义的事情？”
- 抛开人与人之间的隔阂：总结如何更好地测试，并且分享这些测试经验。只有每个人都试图使其所写的代码达到最佳状态时，整体质量才会改进。
- 学习更多关于测试的知识：软件测试行业发展受到各种软件测试创新思维、好的想法的激发。通过参加交流会、加入邮件列表、上网搜索等方式来了解测试前沿发生的事情。
- 学习更多关于开发的东西：参加一种编程培训课程的学习，即使不打算编写大量的代码，也可以把培训当作是在 Bug 领域上的一次侦察飞行。

以上的建议对于测试人员来说是一个启示，测试领域有很多值得探索的东西，有很多值得思考的东西，也有很多值得学习和研究的东西。

### 1.3.3　外包软件测试概述

外包软件测试就是指软件企业将软件项目中的全部或部分测试工作，交给提供软件外包测试服务的公司，由他们为软件进行专门的测试。这样做的好处有两个：一方面，软件企业可以更好地专注核心竞争力业务，同时降低软件项目成本；另一方面，由第三方专业的测试公司进行测试，无论在技术上还是管理上，对提高软件测试的有效性都具有重要意义。

外包软件测试行业前景非常看好，发展空间很大。IDG 的数据显示，最近几年，中国的软件外包产业年均增长率为 36.5%，正处于快速发展的阶段，2008 年预计已达到 16.9

亿美元的市场规模。目前韩日、欧美国家的软件企业纷纷关注中国市场，而作为软件外包强国的印度，在其国内处于前几位的软件外包服务商也准备来“分一杯羹”。从目前市场来看，选择将部分软件测试工作进行外包的公司主要是微软、IBM 等国际软件旗舰企业，他们利用第三方专业软件测试公司，在产品发布前对软件进行一系列的集成测试和系统测试，既保证了测试工作的全面性，又节省了人力、物力的开销。最重要的是，测试结果往往好于这些软件企业最初的预期，效果非常令人满意。软件企业和提供软件外包测试服务的公司进行合作，只要达成双赢，两方皆大欢喜，这样的合作就会越来越多，项目也会越做越大。

### 1.3.4　外包软件测试服务的两种模式

从软件测试服务的组织形式分析，软件外包服务公司有两种常见模式。

第一，在公司内部组建测试实验室和测试部，全部测试都在公司内完成。这是软件本地化企业最希望的模式，便于人员管理和质量控制，但是需要很多软硬件的先期投入。

第二，软件外包服务公司组织测试人员到大型软件开发公司的现场进行测试。这是大多数软件本地化企业不愿意接受却又实际采用最多的模式，主要是因为这种模式便于软件开发公司保证新项目信息保密安全，和监控软件测试的进度和质量。

软件外包测试的兴起意味着更多的机会。通过提供软件测试服务，国内软件本地化公司可以扩展服务业务范围，扩大发展规模，完善内部管理等。

### 1.3.5　外包测试需迈三道坎

对于准备承接软件外包服务的公司而言，要加入外包测试服务队伍，至少需要在三个方面实现跨越：提升国际客户信任度、完善业务流程、招聘大量专业人才。

**1．软件外包三种业务模式**

要了解外包测试服务，需要先了解其业务模式。从为软件开发公司提供外包测试服务的业务模式看，可分为三种类型：现场测试、公司内部测试和设立联合研发中心。

现场测试模式(On-site)是人员外派模式，主要是指外包测试服务公司把自己的人员派到软件开发公司的现场提供服务，这是在做外包服务初期经常采用的一种模式。在这种模式中，外包测试服务公司基本上只提供人员，不控制项目开发的过程，项目实施过程完全由软件开发公司控制。

国内软件外包测试服务公司的内部测试(In-house)模式分为两种：完全离岸外包(Offshore)模式、现场增援与离岸结合的模式(On site+Off shore)。完全离岸外包模式适用于项目比较成熟、定义明确的情况；现场增援和离岸结合起来的模式是指有一些人员要派到欧美软件开发公司那里去，有一些则在中国做测试。比如在美国软件开发公司那边有两三个人的团队，但是在国内可能还有 30 个人、50 个人在一个团队中工作，做同样的项目。

设立联合研发中心使测试外包服务公司同软件开发公司的关系更加紧密，能够深入行业核心业务后采取的模式。这时，双方实际上已经从供应商与服务商关系转化为合作伙伴关系。这种模式在国内出现得比较少，但有逐步朝这个方向发展的趋势。

**2．软件外包测试公司面临三道坎**

软件外包测试属于价值链的低端服务，但低端不等于低技术、低管理、低质量。对于准备承接软件外包服务的公司而言，要加入外包测试服务队伍，至少需要跨越“三道坎”。

第一道坎是难以赢得国际 IT 客户的信赖。中国软件业在空间巨大、利润丰厚的欧美高端市场迟迟未能实现外包突破，这几乎成了很多软件业人士“永远的痛”。目前在软件外包测试方面，虽然这种情况开始有所改变，但要赢得客户信赖，并不是一朝一夕所能达到的。

第二道坎是不完善的业务流程。现代外包测试几乎贯穿软件项目实施的全部过程，包括项目规划、需求分析、方案设计、软件编码和缺陷处理等各个环节，都需要测试者适时介入。由于软件开发存在阶段性和周期性，需要多次对软件中间测试版本(Builds)进行测试。另外，大型软件外包测试需要分布在世界各地的不同公司（软件开发公司、外包测试服务公司等）的项目人员组成一个项目团队，各负其责，并进行有效交流。此外，软件缺陷的报告和修正软件进度报告的提交，软件环境设置、测试工具的选择和测试团队的管理都需要制订科学的流程并得到软件开发公司的认可，以满足国际软件外包测试的要求。

第三道坎是缺乏测试专业人才。外包测试是软件项目实施过程细分的产物，属于为软件开发公司提供技术和质量服务的中间环节。而且软件外包测试是有计划、有组织和有系统的软件质量保证活动，而不是随意的、松散的、杂乱的实施过程，需要符合软件外包测试服务的各类人才包括软件测试执行工程师、测试组长、测试经理，以及市场业务人员共同努力。由于软件外包测试属于新兴职业之一，它通常对从业者的外语能力、学习能力、专注性和职业态度等提出更高的要求，而普通高校和各类社会培训机构以前缺乏这方面的教育课程，因此如何招聘到大量的外包测试人才成为这些外包测试公司面临的棘手问题。

## 1.4　外包软件测试工程师职业素质要求

人是测试工作中最有价值也是最重要的资源，只有保证测试工程师良好的素质，才能保证测试、产品的质量。然而，在软件开发产业中有一种习惯非常普遍，就是让那些经验最少的新手、没有效率的开发者或不适合干其他工作的人去做测试工作。这绝对是一种目光短浅的行为，对一个系统进行有效的测试所需要的技能绝对不比进行软件开发需要的少，事实上，测试者需要获得极其广泛的经验，去解决所遇到许多开发者不可能

遇到的问题。

为高质、高效地完成测试任务，优秀的外包软件测试工程师应具有很好的素质和能力，包括沟通能力、技术能力、自信心、交流能力和幽默感、耐心、很强的记忆力、怀疑一切的精神、勤奋精神、洞察力、适度的好奇心、反向思维和发散思维能力等等。

**1．沟通能力**

优秀的外包软件测试工程师必须具备同测试涉及的所有人进行沟通，不但具有与技术（开发者）和非技术人员（客户、管理人员）的交流，还要与外籍同事进行沟通，因此作为外包测试工程师，要有良好的外语能力。因为测试工作很多都是跨国籍的合作项目，需要大量的沟通和交流，工作中用到的大都是外文文档，如果有语言障碍，就很难顺利完成工作。外包软件测试工程师和用户沟通的重点尽量放在系统可以正确地处理什么和不可以处理什么上，尽量不使用专业术语。对于用户反馈的信息，和开发者或者外籍同事交流时，测试人员须重新组织，以另一种方式表达出来，必须要使用专业术语。测试团队应保证测试小组的成员都能够同用户、开发者和外籍同事同等的交流沟通。

**2．技术能力**

就总体而言，开发人员对那些不懂技术的人持一种轻视的态度。一旦测试小组的某个成员做出了一个比较明显的错误断定，可能会被夸张地到处传扬，那么测试小组的可信度就会受到影响，其他正确的测试结果也会受到质疑。再者，由于软件错误通常依赖于技术，或者至少受构造系统所使用的技术的影响，所以测试人员掌握编程语言、系统构架、操作系统的特性、网络、表示层、数据库的功能和操作等知识，应该了解系统是怎样构成的，明白被测软件系统的概念、技术，要建立测试环境、编写测试脚本，又要会使用软件工程工具。要做到这些，需要有几年以上的编程经验以及对技术和应用领域的深刻理解。

**3．自信心**

开发人员指责测试人员出了错是常有的事，测试工程师必须对自己的观点有足够的自信心，对自己所报的 Bug 有信心。如果没有信心或受开发人员影响过大，测试工作就缺乏独立性，程序中的漏洞或缺陷容易被忽略过去，就谈不上保证软件产品质量。

还有一种情况也是常见的，软件产品设计规格说明书总是或多或少存在一些逻辑问题，编程人员和测试人员对那些有问题的功能存在争议，这时候信心会帮助测试人员发现产品设计中的问题。

**4．外交能力和幽默感**

优秀的测试人员必须能够同测试涉及的所有人（编程、设计等技术人员，客户、管理等非技术人员）进行良好的沟通。机智老练和外交手法有助于维护与开发人员的协作关系，幽默感同样也是很有帮助的。

测试人员应该把精力集中在查找错误上面，而不是放在找出是开发小组中哪个成员引入的错误。这样可以保证测试的否定性结果只是针对产品，而不是针对编程人员，也

就是说要使用一种公正和公平的方式指出具体错误，这对于测试工作是有益的。一般来说，武断地对产品进行攻击是错误的。如果采取的方法过于强硬，对测试者来说，在以后和开发部门的合作方面就相当于“赢了战争却输了战役”。在遇到狡辩的情况下，一个幽默的批评将是很有帮助的。

**5．耐心**

有些软件测试工作需要难以置信的耐心。有时需要花费惊人的时间去分离、识别一个错误，需要对其中一个测试用例运行几十遍，甚至几百遍，了解错误在什么情况，或什么平台下才发生。测试人员需要保持平静，尤其是在集中注意力解决困难问题的时候，特别是在测试执行阶段，面对成百上千个测试用例，要一个个去执行，还要在不同的测试环境上重复，耐心是必要的。当然，我们尽量让测试工具去完成那些重复性的任务。

**6．很强的记忆力**

一个优秀的测试工程师应该有能力将以前曾经遇到过的类似错误从记忆深处挖掘出来，这一能力在测试过程中的价值是无法衡量的。因为许多新出现的问题和已经发现的问题相差无几。

在测试一个产品的高版本时，对以前所发布的各种版本产品功能清楚，就很容易了解新版本的功能做了哪些改动、为什么那么改、改了之后会对其他特性有哪些影响等一系列问题。如果熟悉软件各种老版本所出现的缺陷，有助于对新版本的用例设计和测试执行。

**7．怀疑精神**

可以预料，开发人员会尽他们最大的努力将所有的错误解释过去，测试人员必须听每个人的说明，但必须保持高度警惕、怀疑一切，直到自己的分析结果或亲自测试之后，才做出决定。并具有自我督促能力，才能够保证每天的工作都能高质量完成。

**8．洞察力**

一个好的测试工程师具有一种先天的敏感性，并且还能尝试着通过一些巧妙的变化去发现问题。同时，还具有“测试是为了破坏”的观点，捕获用户观点的能力，强烈的质量追求，对细节的关注能力。应用的高风险区的判断能力以便将有限的测试针对重点环节。

**9．适度的好奇心**

优秀的测试工程师在开发测试用例时使用的方法，与勘探专家在一个山洞中摸索前进的方法一样。虽然周围可能存在大量的死路，但是测试工程师具有适度的好奇心，会促使他们向山洞中的深处探索，向一切没有去过的地方前进，最终可能会有一个大发现。

编写出导致错误出现的测试用例，这就需要好奇心。测试工程师必须阅读规格说明，与开发人员一起讨论“假设分析”的场景，并在大脑中反复思考被测试系统，还要从所有的角度加以检查。测试工程师如果没有好奇心并对要达到的目标缺乏强烈兴趣，那么他只能写出肤浅的测试用例。

如果测试人员在一个错误上花费很多时间，通过尝试很多变体去探索造成这种错误出现的根本原因，这样做也是不正确的。所以，好奇心需要适度。应该使用什么标准去区分“足够好奇”、“不够好奇”和“过分好奇”呢?在及时完成测试执行任务和编写灵活高效的测试用例之间，在进度的压力和探究错误发生根源之间，优秀的测试人员能够取得平衡。

**10．反向思维和发散思维能力**

测试工程师应想尽办法来考虑产品可能出现失败的各种方式会最大限度地暴露其存在的问题、用严格的边界条件来检验它，让系统经受压力测试，或者是强迫它处理“不可能发生的”错误。因此，优秀的测试人员应具有是在一种安全的环境下发现错误，并且之后可以让项目小组来修复它们。测试需要通过悲观的思想倾向去追求崇高的目标，只有提高产品的质量才能使公司取得成功。

# 第 2 章　软件与软件测试的概述

**学习目标：**

1．了解什么是软件测试
2．了解软件测试的起源
3．掌握软件测试的定义
4．了解软件测试模型

## 2.1　软件和软件开发

### 2.1.1　软件的含义

我们每天都说软件测试、软件开发……但是否真正全面理解什么是软件，大多数人不敢肯定。那软件真正的含义是什么？

先看看一般教科书所给出的规范、科学的定义，即软件是：

- 能够完成预定功能和性能的、可执行的指令（计算机程序）。
- 使得程序能够适当地操作信息的数据结构。
- 描述程序的操作和使用的文档。

即“软件=程序+数据（库）+文档”，图 2-1 给出了软件的最基本的组成成分。实际上，还少了一项内容——服务。我们可以用一个简单的公式给出软件的定义：

软件=程序+数据（库）+文档+服务

为了帮助学员更好地理解软件的含义，我们一起来看看软件有哪些特征。软件是相对硬件而存在的。硬件是可以直观感觉到、触摸到的物理产品。生产硬件时，人的创造性的过程（设计、制作、测试）能够完全转换成物理的形式。例如，生产一个新的计算机，从初始的草图、正式的设计图纸和面板的原型，一步一步演化成为一个物理的产品，如模具、集成芯片、集成电路、电源和塑料机箱等。

正如我们政府官员经常提到的，我们不仅要搞好投资的硬件环境，更要搞好软件环境。这里的硬件环境，包括交通、水电、办公楼和厂房等，而软件环境指的就是优惠政策、政府职能转变和服务等。

软件则是逻辑的、知识性的产品集合，是对物理世界的一种抽象，或者是某种物理形态的虚拟化。因此，软件具有与硬件完全不同的特征。其主要表现在以下 3 个方面。




图 2-1　软件的概念

**1．软件是硬件的灵魂，硬件是软件的基础**

计算机硬件必须靠软件实现其功能，如果没有软件，硬件就好比一堆废铁，所以说软件是硬件的灵魂。同时，软件必须依赖于硬件，只有在特定的硬件环境上才能运行。

虽然“软件工厂”的概念也已被引入，这并不是说硬件生产和软件开发是一回事，而是引用软件工厂这个概念促进软件开发中模块化设计、组件复用等意识的全面提升。

**2．软件是智慧和知识的结晶**

软件是完全的智力产品，是通过技术员的大脑活动创造的结果。软件现在被认为属于高科技产品。软件产业是一种知识密集型产业。

一个价值很高的软件，可能就装在几张软盘上，包括程序和文档。少数不了解软件价值的领导，不愿意为此付出几十万元人民币。他可能会说，几十万元钱可以买一大堆计算机，可以买一辆桑塔纳或奥迪小轿车，几张软盘哪会值那么多钱？在这里，这位领导只看到了软件的载体，也就是只看到其物理的表现形式，而没有看到其实质的内容以及开发这个产品过程中所投入的、高技术的大量人力。软件的主要成本在于先期的开发人力。软件成为产品之后，其后期维护、服务成本也很高。而软件载体的制作成本很低，如磁盘、光盘的复制是比较简单的，所以软件也就容易成为盗版的主要目标。

**3．软件不会“磨损”，而是逐步完善**

随着时间的推移，硬件构件会由于各种原因受到不同程度的磨损，但软件不会。新的硬件故障率很低，随着长时间的改变，硬件会老化，故障率会越来越高。相反，隐藏

的错误会引起程序在其生命初期具有较高的故障率，随着使用的不断深入，所发现的问题会慢慢地被改正，其结果是程序越来越完善，故障率会越来越低。

从另一个侧面看，硬件和软件的维护差别很大。当一个硬件构件磨损时，可以用另外一个备用零件替换它，但对于软件，不存在替换，而是通过开发补丁程序不断地解决适用性问题，或扩充其功能。一般来说，软件维护要比硬件维护复杂得多，而且软件的维护周期要长得多。软件正是通过不断的维护，改善功能，增加新功能，来提高软件系统的稳定性和可靠性的。

### 2.1.2　开发中的人员角色

随着行业的快速发展，软件开发环境变得越来越复杂。为此，IBM 推出了全新的软件开发商业流程理念——SDP（Software Development Platform，软件开发平台），使得原有的“生命周期开发”得到进一步完善，从而在一个高度集成、整合的环境下，为用户提供高质量的软件开发解决方案。

众所周知，软件开发是一个需要高度协作的技术过程。开发人员根据自身的特点，在开发过程中扮演的角色也不尽相同。SDP 把整个开发人员群体定义为 6 种角色，即分析人员、架构师、数据设计人员、项目经理、开发人员和系统测试人员。这个定义明确了软件开发过程中不同角色的定位，从而大幅提升了软件开发的管理水平，更为扮演不同角色的开发人群指明了相应的发展方向。与此同时，Rational 软件还为 6 种开发群体准备了相应的技术和产品支持。真正做到根据不同角色软件开发人员的定位和需要，进行“量身打造”。充分体现了其人性化的特点。

**1．分析人员**

业务分析人员的任务是理解和描绘客户的需求，引导和协调用户和业务需求的收集和确认，文档化和组织系统的需求，或者向整个团队传达需求。

**2．架构师**

架构师负责理解系统的业务需求，并创建合理、完善的系统体系架构。架构师也负责通过软件架构来决定主要的技术选择。这典型的包括识别和文档化系统的重要架构方面，包括系统的需求、设计、实现和部署“视图”。

**3．数据设计人员**

对于大多数的应用开发项目来说，用于持久存储数据的技术是关系型数据库。数据库架构师负责定义详细的数据库设计，包括表、索引、视图、约束、触发器、存储过程和其他的特定数据库用于存储、返回和删除持久性对象的结构。

**4．项目经理**

项目经理负责管理业务应用开发或者软件和系统开发项目。项目经理角色计划、管理和分配资源，确定优先级，协调用户和客户的交互。项目经理也要建立一系列的实践活动以确保项目工作产品的完整性和质量。

**5．开发人员**

开发人员通常负责设计和实现可执行的代码方案、测试开发出的组件和分析运行时的情况以去除可能存在的错误。有时开发人员还负责创建软件的体系架构或者使用快速应用开发工具。

**6．系统测试人员**

系统测试人员负责制订测试计划并依照测试计划进行测试。这些测试包括功能性的测试（黑盒测试）和非功能性的测试（白盒测试）。测试人员需要良好的测试工具来辅助完成测试任务，自动化的测试工具将大幅度提高测试人员的工作效率和质量。

## 2.1.3 软件开发瀑布模型

软件开发的基本过程，可以被简单地分为需求分析、软件设计（概要设计、详细设计）、编程、测试和维护等几个阶段，即通常所说的“传统生命周期”，也就是著名的软件开发过程的“瀑布模型”，如图 2-2 所示。通过这个模型，能比较直观地理解软件开发的全过程。

1970 年 Winston Royce 提出了著名的“瀑布模型”，直到 20 世纪 80 年代早期，它一直是唯一被广泛采用的软件开发模型。

瀑布模型将软件生命周期划分为需求分析、软件设计、程序编写、软件测试和运行维护 5 个基本活动，并且规定了它们自上而下、相互衔接的固定次序，如同瀑布流水，逐级下落。

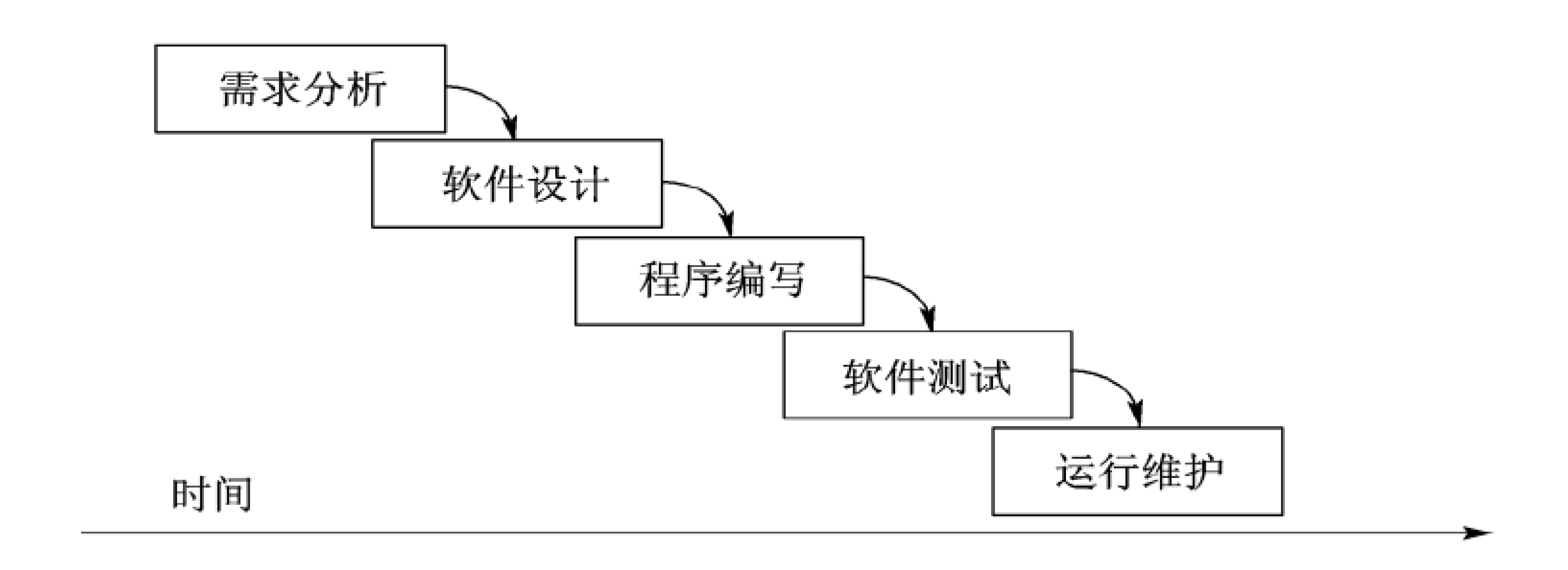



图 2-2 软件开发过程的瀑布模型

**1．需求分析**

需求分析是根据客户的要求，清楚地了解客户需求中的产品功能、特性、性能、界面和具体规格等，然后进行分析，确定软件产品所能达到的目标。软件产品需求分析是软件开发过程的第一个环节，也是最重要的一个环节。如果需求分析做不好，下面的设计、编程做得再好，客户（用户）也不可能对开发出来的软件产品感到满意。软件产品需求分析的结果要文档化，如 MRD(Marketing Requirement Document)，而且这类文档的

描述尽量不要用专业术语，从而使用户能够完全理解需求分析的结果，参与对其复审的过程。

**2．设计**

软件设计是根据需求分析的结果，考虑如何在逻辑、程序上去实现所定义的产品功能、特性等。可以分为概要设计和详细设计，也可以分为数据结构设计、软件体系结构设计、应用接口设计、模块设计、算法设计、界面设计等。设计过程将需求转换成软件表示，设计的结果将作为编码的框架和依据，以提高编码的效率和质量。设计的文档化体现在产品规格说明书(Functional Specification)、技术设计文档(Development Design Document)和软件配置文档(Software Configuration Document)。

**3．编程**

经过需求分析、设计之后，接下来就是用一种或多种具体的程序语言（如 C/JSP 等）进行编码，即将设计转换成计算机可读的形式。如果设计做得好、做得仔细，编程就容易了。

**4．测试**

任何编程，免不了存在这样或那样的错误，所以有必要进行软件测试。测试过程集中于软件的内部逻辑——保证所有语句都测试到，以及外部功能——引导测试去发现错误，并保证定义好的输入能够产生与预期结果相同的输出。测试阶段按不同的过程阶段分为单元测试、集成测试、系统测试、验收测试等。

**5．维护**

从理论上，软件测试的覆盖率不可能做到百分之百，所以软件在交付给用户之后有可能存在某些问题，而且用户的需求会发生变化，特别是开始使用产品之后，对计算机系统有了真正的认识和了解，会提出适用性更好的、功能增强的要求。所以，软件交付之后不可避免地要进行修改、升级等。正如前面所说，软件维护复杂、周期长，其成本必然很高。通过提高软件的需求分析、设计和编程的质量，强化软件测试，可以大幅度降低软件的维护成本。

### 2.1.4　软件可靠性

可靠性（Reliability）是产品在规定的条件下和规定的时间内完成规定功能的能力，它的概率度量称为可靠度。

软件可靠性（Software Reliability）是软件系统固有特性之一，它表明了一个软件系统按照用户的要求和设计的目标，执行其功能的正确程度。软件可靠性与软件缺陷有关，也与系统输入和系统使用有关。理论上说，可靠的软件系统应该是正确、完整、一致和健壮的。但是实际上任何软件都不可能达到百分之百的正确，而且也无法精确度量。一般情况下，只能通过对软件系统进行测试来度量其可靠性。

软件可靠性给出如下定义："软件可靠性是软件系统在规定的时间内及规定的环境条

件下，完成规定功能的能力。”根据这个定义，软件可靠性包含了以下 3 个要素：

（1）规定的时间。

软件可靠性只是体现在其运行阶段，所以将“运行时间”作为“规定的时间”的度量。“运行时间”包括软件系统运行后工作与挂起（开启但空闲）的累计时间。由于软件运行的环境与程序路径选取的随机性，软件的失效为随机事件，所以运行时间属于随机变量。

（2）规定的环境条件。

环境条件指软件的运行环境。它涉及软件系统运行时所需的各种支持要素，如支持硬件、操作系统、其他支持软件、输入数据格式和范围以及操作规程等。不同的环境条件下软件的可靠性是不同的。具体地说，规定的环境条件主要是描述软件系统运行时计算机的配置情况以及对输入数据的要求，并假定其他一切因素都是理想的。有了明确规定的环境条件，还可以有效判断软件失效的责任在用户方还是研制方。

（3）规定的功能。

软件可靠性还与规定的任务和功能有关。由于要完成的任务不同，软件的运行情况会有所区别，则调用的子模块就不同（即程序路径选择不同），其可靠性也就可能不同。所以要准确度量软件系统的可靠性必须首先明确它的任务和功能。

在讲到软件可靠性评估的时候，我们不得不提到软件可靠性模型。软件可靠性模型（Software Reliability Model）是指为预计或估算软件的可靠性所建立的可靠性框图和数学模型。建立可靠性模型是为了将复杂系统的可靠性逐级分解为简单系统的可靠性，以便于定量。

### 2.1.5　软件缺陷产生的原因

由于软件开发人员思维上的主观局限性，且目前开发的软件系统都具有相当的复杂性，决定了在开发过程中出现软件错误是不可避免的，软件过多的或严重的错误会导致程序或系统的失效。很多资料总结出软件错误产生的主要原因有以下几点：

- 需求规格说明书(Requirement Specification 或 Functional Specification)包含错误的需求，或漏掉一些需求，或没有准确表达客户所需要的内容。
- 需求规格说明书中有些功能不可能或无法实现。
- 系统设计(System Design)中的不合理性。
- 程序设计中的错误。
- 程序代码中的问题，包括错误的算法、复杂的逻辑等。
- 若能及早排除软件开发中的错误，有效地减少后期工作可能遇到的问题，就可以尽可能地避免付出高昂的代价，从而大大提高系统开发过程的效率。

在实际的软件测试项目中缺陷产生的原因还避免不了以下几种情况：

- 项目期限的压力。

- 产品的复杂度。
- 沟通不良。
- 开发人员疲劳、压力过大或受到干扰。
- 不了解客户的需求。
- 缺乏动力。

我们可以从软件自身特点、团队工作和项目管理等多个方面进行分析，找出导致软件缺陷更多的一些原因，这可以归纳为如下 3 个方面。

**1. 软件开发过程自身的特点造成的问题**

- 软件需求定义难以做到清清楚楚，导致设计目标偏离客户的需求，从而引起功能或产品特性上的缺陷。
- 软件系统结构非常复杂，而又无法构造一个完美的层次结构或组件结构，结果将导致意想不到的问题。
- 新技术的采用，可能涉及技术或系统兼容性的问题，而事先没有考虑到。
- 对程序逻辑路径或数据范围的边界考虑不周全，容易在边界条件上出错，或者超出边界条件后又缺少保护导致出错。
- 没有考虑或处理好系统崩溃后的自我恢复、故障转移或数据的异地备份等情况，从而存在系统安全性、可靠性的隐患。

**2. 软件项目管理的问题**

- 受质量文化的影响，不重视质量计划，对质量、资源、任务、成本等的平衡性把握不好，容易挤掉需求分析、评审、测试等的时间，于是遗留的缺陷也会比较多。
- 开发周期短，需求分析、设计、编程、测试等各项工作不能完全按照定义好的流程来进行，工作不够充分，结果也就不完整、不准确，错误较多；周期短，还给各类开发人员造成太大压力，从而引起一些人为的错误。
- 开发流程不够完善，存在较多的随机性和缺乏严谨的内审和评审机制，容易产生问题。
- 文档不完善，风险估计不足。

**3. 团队工作的问题**

- 沟通不够、不流畅，导致不同阶段、不同团队的开发人员对问题的理解不一致。
- 项目组成员技术水平参差不齐，或者新员工较多或培训不够等，也容易引起问题。

## 2.2　软件测试的起源

通常称之为 Bug 的软件缺陷是伴随着软件出现的，而软件测试同样是伴随着软件的出现而出现，并且随着软件的 Bug 日益增多，造成了日益严重的质量事故。因此，人们“对抗”Bug 的态度日益坚决，使得软件测试不断地得到加强、重视和持续发展。

第一个 Bug 的故事。故事发生在 1945 年 9 月的某一天，在一间老式建筑的窗户外面飞进来一只飞蛾，Hopper 正埋头工作在一台名为 Mark Ⅱ的计算机前，没有注意到这只即将造就历史事件的飞蛾。这台计算机使用了大量的继电器（电子机械装置，那时还没有使用晶体管）。

突然，Mark Ⅱ死机了。Hopper 试了很多次还是不能启动，Hopper 开始用各种方法查找问题，看问题究竟出现在哪里，最后 Hopper 确定是某个电路板的继电器出错了。Hopper 观察这个出错的继电器，惊奇地发现一只飞蛾躺在里面。Hopper 小心地用镊子将飞蛾夹出来，用透明胶布粘到“事件记录本”中，写上“第一个发现虫子的实例”。

Hopper 的事件记录本，连同那只飞蛾，现在都陈列在美国历史博物馆中。如图 2-3 所示的照片就是那只飞蛾以及 Hopper 的记录。

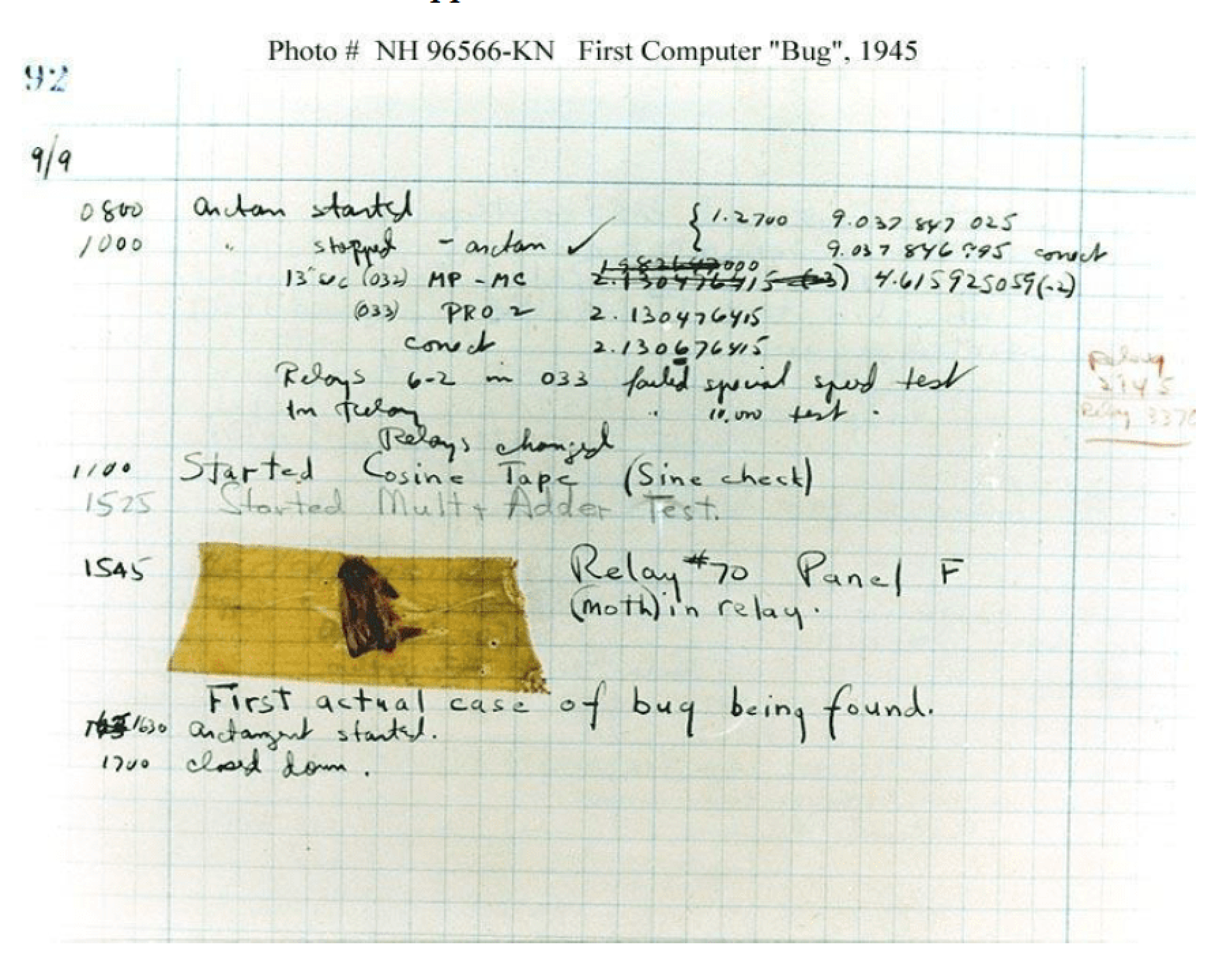



图 2-3 Hopper 关于飞蛾事件的记录

## 2.3 软件测试的重要性

现今软件无处不在，人们在不同的场合都有可能会不知不觉地使用软件，如日常生活中的手机、智能冰箱、新一代的数字彩电、计算机等。在日常使用软件中，也或多或少会碰到一些不愉快的事情，如信号显示不对、数据不完整、操作不灵活等，但软件问

题有时引起的麻烦还不止这些，有时造成的危害可能会非常严重。这一节，通过一系列典型的软件质量问题实例，阐述一个简单又重要的道理——软件测试的重要性。

### 2.3.1 软件缺陷带来的教训

**1．千年虫**

在 20 世纪 70 年代，程序员为了节约非常宝贵的内存资源和硬盘空间，在存储日期时，只保留年份的后两位，如“1980”被存为“80”。但是，这些程序员万万没有想到他们的程序会一直被用到 2000 年，当 2000 年到来的时候，问题就会出现。比如银行存款程序在计算利息时，应该用现在的日期“2000 年 1 月 1 日”减去当时存款的日期，比如“1989 年 1 月 1 日”，结果应该是 21 年，如果利息是 3%，每 100 元银行要付给顾客大约 86 元利息。如果程序没有纠正年份只存储两位，其存款年数就变为–89 年，变成顾客反要付给银行 1288 元的巨额利息。所以，当 2000 年快要来到的时候，为了这样一个简单的设计缺陷，全世界付出几十亿美元的代价。

**2．爱国者导弹防御系统**

美国爱国者导弹防御系统是主动战略防御（即星球大战）系统的简化版本，它首次被用在第一次海湾战争对抗伊拉克飞毛腿导弹的防御作战中，总体上看效果不错，赢得各界的赞誉。但它还是有几次失利，没有成功拦截伊拉克飞毛腿导弹，其中在沙特阿拉伯的多哈爆炸的一枚飞毛腿导弹造成 28 名美国士兵死亡。分析专家发现，拦截失败的症结在于一个软件缺陷，当爱国者导弹防御系统的时钟累计运行超过 14 小时后，系统的跟踪系统就不准确。在多哈袭击战中，爱国者导弹防御系统运行时间已经累计超过 100 多个小时，显然那时系统的跟踪系统已经很不准确，从而造成这种结果。

**3．迪斯尼的圣诞节礼物**

1994 年圣诞节前夕，迪斯尼公司发布了第一个面向儿童的多媒体光盘游戏“狮子王童话”。尽管在此之前，已经有不少公司在儿童计算机游戏市场上运作多年，但对迪斯尼公司而言，还是第一次进军这个市场。由于迪斯尼公司的著名品牌和事先的大力宣传及良好的促销活动，结果，市场销售情况非常不错，该游戏成为父母为自己孩子过圣诞节的必买礼物。

但结果却出人意料，12 月 26 日，圣诞节后的第一天，迪斯尼公司的客户支持部电话开始响个不停，不断有人咨询、抱怨为什么游戏总是安装不成功，或没法正常使用。很快，电话支持部门就淹没在愤怒家长的责问声和玩不成游戏孩子们的哭诉之中，报纸和电视开始不断报道此事。

后来证实，迪斯尼公司没有对当时市场上的各种 PC 机型进行完整的系统兼容性测试，只是在几种 PC 机型上进行了相关测试。所以，这个游戏软件只能在少数系统中正常运行，但在大众使用的其他常见系统中却不能正常安装和运行。

2007 年 5 月 18 日，诺顿杀毒软件升级病毒库后，会把 Windows XP 系统的关键系

统文件当作病毒清除，重启后系统将会瘫痪。截至中午 12 点已有超过 7 千名个人用户和近百家企业用户向瑞星客户服务中心求助，更多用户由于系统繁忙无法打入电话。瑞星安全专家表示，安装了 MS06-070 补丁的 XP 系统，如果将诺顿升级最新病毒库，则诺顿杀毒软件会把系统文件 netapi32.dll、lsasrv.dll 隔离清除，从而造成系统崩溃。由于国外品牌的笔记本和台式机多数预装了 Windows XP 系统和诺顿杀毒软件，这些用户极其容易遭到此次“误杀”攻击，因此中国大陆地区将有数百万台计算机面临崩溃的危险。由于该次误杀只发生在简体中文版的 XP 系统上，因此对国外用户几乎没有影响。瑞星客户服务中心的疫情监控工程师分析，赛门铁克在企业级市场上占有相当大的份额，特别在金融、电信等行业拥有一定的优势，因此此次误杀会导致许多企业网络完全瘫痪。

我们看到了当软件失败时发生的历史事件。后果也许仅是不方便，比如计算机不能正常运行，也可能导致灾难性的后果，比如致人死命。在这些事件中，显然软件没有按照预期目标运转。作为软件测试员，可能所发现的大多数问题不是那么明显、严重，而是难以觉察的简单而细微的错误，有些是真正的错误，也有些不是。除了上述一些实例外，还有很多由于软件缺陷而造成的事例，它们同样会给人们带来警示，不要忽视软件的缺陷。

### 2.3.2 测试是软件开发的重要环节之一

在 G.J.Myers 的经典著作《软件测试的艺术》中给出了测试的定义：“程序测试是为了发现错误而执行程序的过程。”测试的目的是发现程序中的错误，是为了证明程序有错，而不是证明程序无错。在软件开发过程中，分析、设计与编码等工作都是建设性的，唯独测试似乎带有“破坏性”。测试可视为分析、设计和编码 3 个阶段的“最终复审”，在软件质量保证中具有重要地位。为了确保软件的质量，较理想的做法应该是对软件的开发过程，按软件工程各阶段形成的结果，分别进行严格的审查。

软件测试在软件生命周期中占据重要的地位，在传统的瀑布模型中，软件测试学仅处于运行维护阶段之前，是软件产品交付用户使用之前保证软件质量的重要手段。近来，软件工程界趋向于一种新的观点，即认为软件生命周期每一阶段中都应包含测试，从而检验本阶段的成果是否接近预期的目标，尽可能早地发现错误并加以修正，如果不在早期阶段进行测试，错误的延时扩散常常会导致最后成品测试的巨大困难。

事实上，对于软件来讲，不论采用什么技术和什么方法，软件中仍然会有错。采用新的语言、先进的开发方式、完善的开发过程，可以减少错误的引入，但是不可能完全杜绝软件中的错误，这些引入的错误需要测试来找出，软件中的错误密度也需要测试来进行估计。测试是所有工程学科的基本组成单元，是软件开发的重要组成部分。自有程序设计的那天起测试就一直伴随着。统计表明，在典型的软件开发项目中，软件测试工作量往往占软件开发总工作量的 40%以上。而在软件开发的总成本中，用在测试上的开

销要占 30%～50%，如果把维护阶段也考虑在内，讨论整个软件生存期时，测试的成本比例也许会有所降低，但实际上维护工作相当于二次开发，乃至多次开发，其中必定还包含有许多测试工作。

软件测试在产品开发中占据相当重要的一部分，是软件行业二十几年的实践所证明的一个道理，或者说是从不断的失败中总结出来的经验。以微软公司为例，大家可以感觉到，微软以前的产品时时会发生崩溃、死机等现象，而今天的产品则比 5 年前的产品功能要强大得多，稳定性却好得多。为什么呢?这是因为微软公司重视测试工作，测试人员越来越多，如今微软的软件测试人员是开发人员的 1.5～2.5 倍。其次，测试人员越来越有经验，测试工作也就越做越好。正是由于清晰地认识到了软件测试的重要性，微软的产品质量才有了明显的提高。

最初，微软公司与大家一样，认为测试不重要，重要的是开发人员。通常，一个团队中有几百个开发人员，但只有几个测试人员，并且开发人员的待遇要比测试人员高很多。经过多年的实践后，微软公司发现，去修正那些出现问题的产品所花的钱，比多聘用几个测试人员的费用要多得多，所以，开始不断地聘用测试人员。同时，现在测试人员的待遇和开发人员的待遇非常接近，测试人员水平越高，找到 Bug 时间就越早，软件就越容易更正，产品发布之后越稳定，公司赚的钱也越多。这也是多数软件公司慢慢悟出来的道理，软件测试是软件产品开发中最重要的几个环节之一。

## 2.4　什么是软件测试

### 2.4.1　软件测试的定义

在工业制作和生产中，测试被当作一个常规的检验产品质量的生产活动。测试的含义为“以检验产品是否满足需求为目的”。而软件测试活动包括了很重要的任务，即发现错误。

那究竟什么是软件测试呢？软件测试的定义是什么呢？1983 年 IEEE 提出的软件工程标准术语中给软件测试下的定义是:**“使用人工或自动手段来运行或测定某个系统的过程，其目的在于检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差别。”**这就非常明确地提出了软件测试以检验是否满足需求为目标。

我们知道软件是由文档、数据以及程序组成的，那么软件测试就应该是对软件形成过程的文档、数据以及程序进行的测试，而不仅仅是对程序进行的测试。

随着人们对软件工程化的重视以及软件规模的日益扩大，软件分析、设计的作用越来越突出，并且也有相关的资料表明，60%以上的软件程序错误并不是程序错误，而是分析和设计的错误。因此，我们要做好软件需求和设计阶段的测试工作，就相当重要，

这就是我们提倡的测试概念扩大化，提倡软件全生命周期测试的理念。

### 2.4.2 软件测试的目的

我们已经讲到了什么是软件测试，那么软件测试的目的是什么呢？这是一个看起来很简单、不太值得讨论的问题，但往往这样的问题其实是很难回答的。

早期的软件定义指出软件测试目的是寻找错误，并且尽最大的可能找出最多的错误。

G. J. Myers 就软件测试的目的提出了以下的观点。

- 测试是为了证明程序有错，而不是证明程序无错误。
- 一个好的测试用例是在于它能发现以前未发现的错误。
- 一个成功的测试是发现了以前未发现的错误的测试。

软件测试是程序的一种执行过程，目的是尽可能发现并改正被测试软件中的错误，提高软件的可靠性。它是软件生命周期中一项非常重要且非常复杂的工作，对软件可靠性保证具有极其重要的意义。在目前形式化方法和程序正确性证明技术还无望成为实用性方法的情况下，软件测试在将来相当一段时间内仍然是软件可靠性保证的有效方法。软件工程的总目标是充分利用有限的人力和物力资源，高效率、高质量地完成软件开发项目。不足的测试势必使软件带着一些未揭露的隐藏错误投入运行，这将意味着更大的危险让用户承担。过度测试则会浪费许多宝贵的资源。到测试后期，即使找到了错误，然而付出了过高的代价。E.W.Dijkstra 的一句名言说明了这一道理：“程序测试只能表明错误的存在，而不能表明错误不存在。”可见，测试是为了使软件中蕴涵的缺陷低于某一特定值，使产出、投入比达到最大。

### 2.4.3 软件测试的原则

根据软件测试的目的，软件测试的原则如下。

**1．应当把“尽早地和不断地进行软件测试”作为软件开发人员的座右铭**

由于软件所涉及问题的复杂性、软件本身的复杂性和抽象性、软件开发各个阶段工作的多样性，以及参加开发各种层次人员之间工作的配合关系等因素，使得开发的每个环节都可能产生错误。所以人们不应把软件测试仅仅看作是软件开发的一个独立阶段，而应当把它贯穿到软件开发的各个阶段中。坚持在软件开发的各个阶段进行技术评审，这样才能在开发过程中尽早发现和预防错误，把出现的错误克服在早期，杜绝某些发生错误的隐患，减少开发费用，提高软件质量。

越早发现错误，则修改的代价越小。越迟发现错误，修复软件需要付出的代价就越高，如图 2-4 所示。

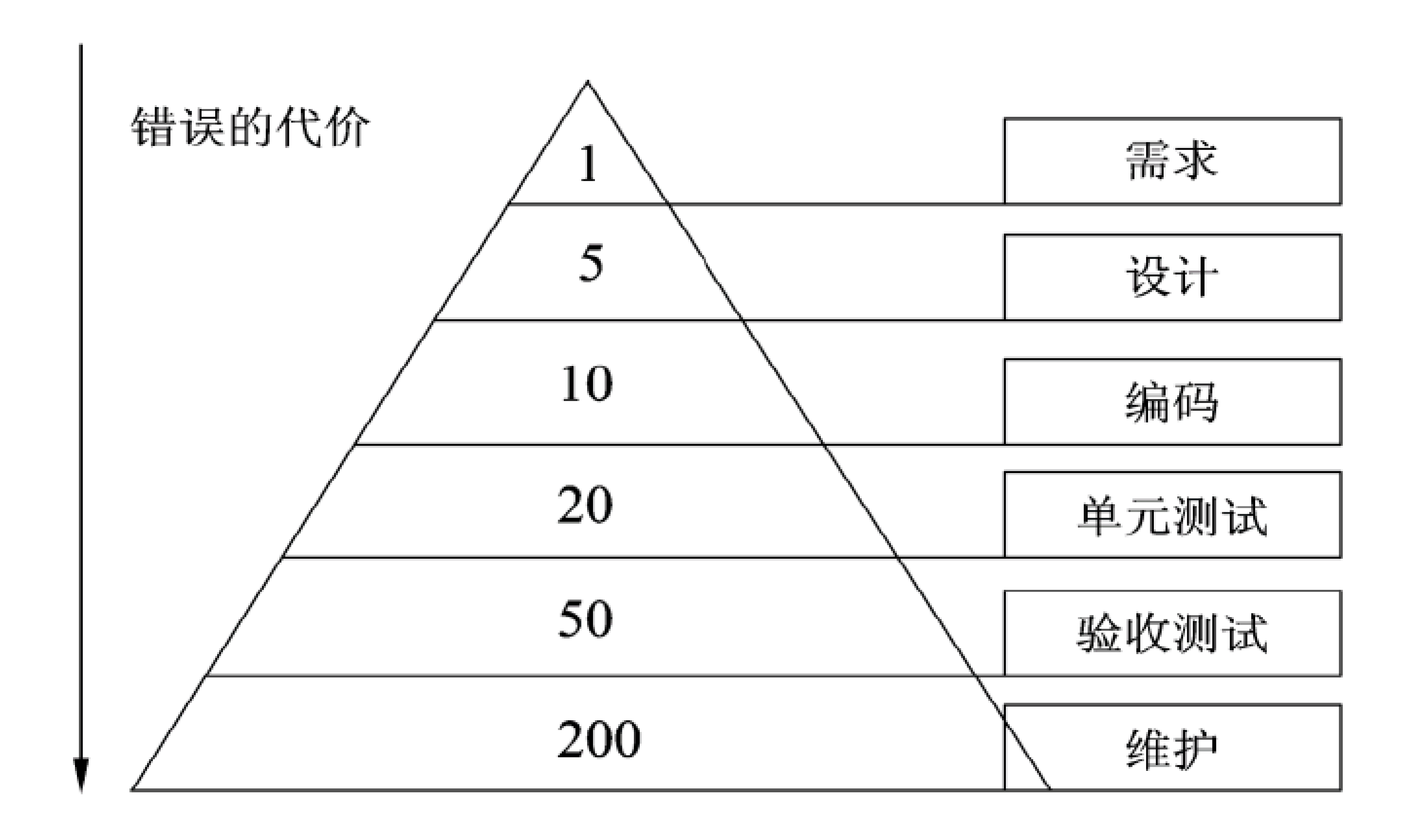



图 2-4　在不同阶段修改错误的代价

**2．Good Enough 原则**

Good Enough 原则是指测试的投入与产出要适当权衡，测试进行得不够充分是对质量不负责任的表现，但是投入过多的测试，则是资源浪费的表现。软件测试的投入与产出关系如图 2-5 所示。

随着软件测试的投入，测试的产出随之增加；但是当测试投入增加到一定的比例后，其测试效果并不会明显地增加。

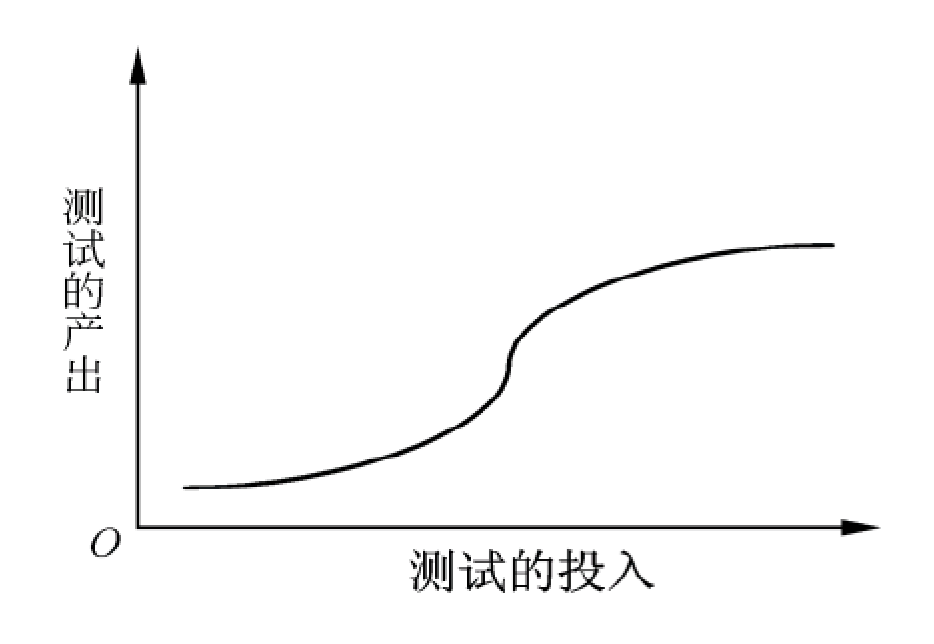



图 2-5　测试的投入产出关系

如果在一个测试项目中盲目地增加测试资源，如测试人员、测试工具等，并不一定能带来更高的效率和更大的收益。因为增加人员的同时，可能会增加沟通与培训的成本。增加工具则可能带来学习和培训的成本。尤其是在进度比较紧迫的测试项目中，为了加快测试进度而盲目地增加资源，可能会带来相反的结果。

零缺陷是理想的追求，而 Good Enough 则是现实的追求。不能盲目追求最佳的测试效果而投入过多的测试资源，应该根据项目实际要求和产品的质量要求来考虑测试的投入。

技巧：适当加入其他的质量保证手段，例如代码审评、同行评审、设计评审等，可以有效地降低对测试的依赖，并且确保软件缺陷能尽早被发现，从而降低总体成本。

**3. Pareto 原则**

Pareto 原则，也叫 80-20 原则，是 1879 年由意大利人 Villefredo Pareto 提出的。Pareto 原则的原义是：社会财富的 80%是掌握在 20%的人手中，而余下的 80%的人只占有 20%的财富。后来，这种“关键的少数和次要的多数”的理论被广泛应用在社会学和经济学中，并称之为 Pareto 原则。在软件测试中的 80-20 原则是指 80%的 bug 在分析、设计、评审阶段就能被发现和修正，剩下的 16%则需要由系统的软件测试来发现，最后剩下的 4%左右的 bug 只有在用户长时间的使用过程中才能暴露出来，如图 2-6 所示。

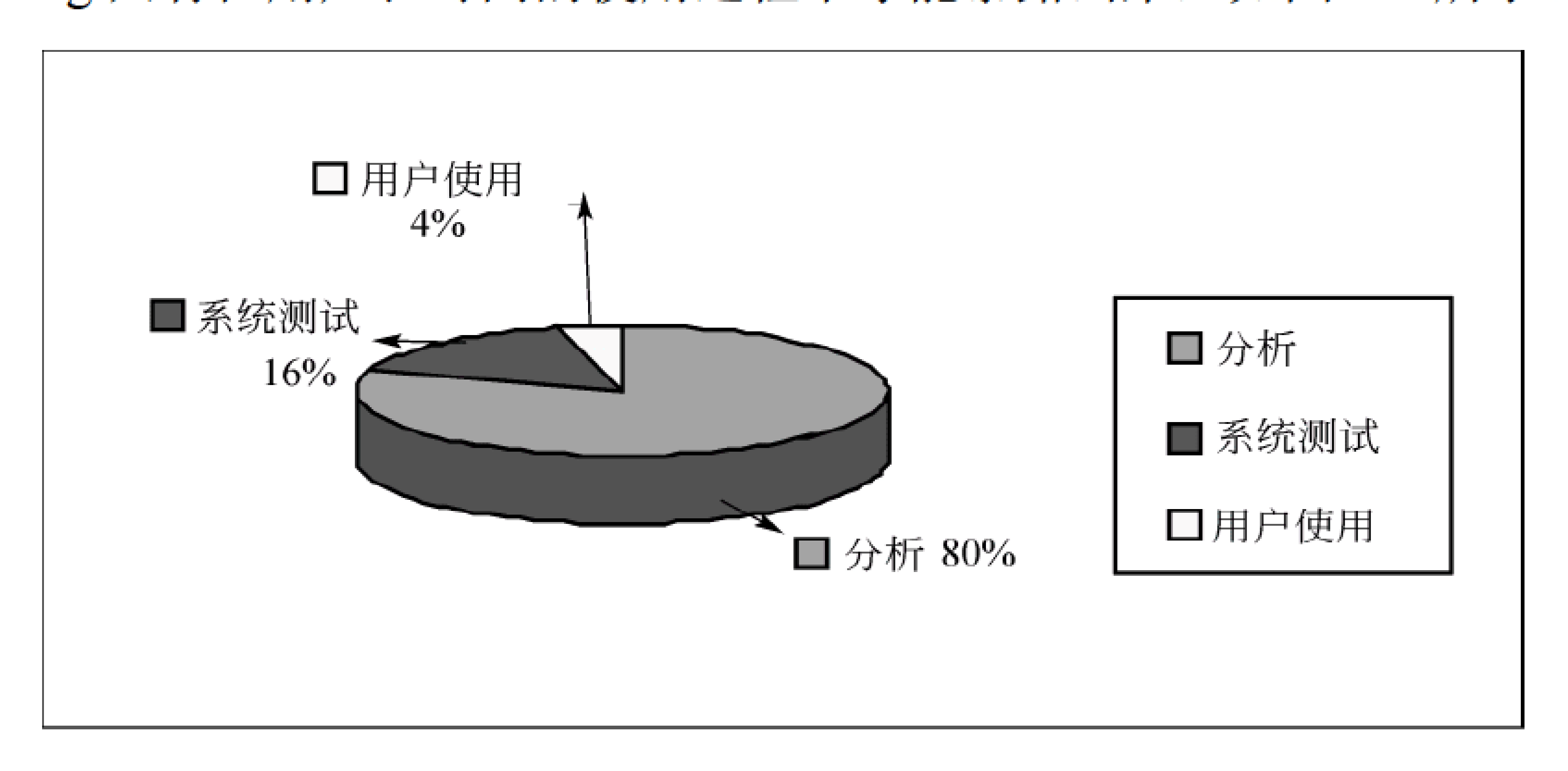



图 2-6 软件缺陷的发现时机分布

基本上可以根据图中分布来定义缺陷逃逸率，即多少缺陷在发布前未被测试到，而是逃逸到了用户的手中。对于一个广泛使用的程序，其维护成本通常是开发成本的 40%以上，并且维护成本受用户数量的严重影响。用户越多，所发现的缺陷也越多。

注意：测试不能保证发现所有的错误，但是测试人员应该尽可能多地发现错误，避免应该在开发阶段发现的错误逃逸到用户那里。

程序员应尽可能避免测试自己编写的程序，程序开发组也应尽可能避免测试本组开发的程序。如果条件允许，最好建立独立于开发组和客户的第三方测试组或测试机构。

但这并不是说程序员不能测试自己的程序，而是说由别人来测试可能会更客观、更有效，并更容易取得成功。要注意的是，这点不能与程序的调试（Debugging）相混淆。调试由程序员自己来做可能更有效。

**4. 充分注意测试中的群集现象**

“物以类聚”，软件缺陷也同样有聚集效应。软件缺陷的聚集通常是由缺陷出现的阶

段时间、程序员的开发状态，或者是缺陷出现的代码范围的复杂度导致的。

测试时不要被一开始发现的若干错误所迷惑，找到了几个错误就以为问题已经解决，不需要继续测试了。经验表明，测试后程序中残存的错误数目与该程序中已发现的错误数目成正比，如图 2-7 所示。根据这个规律，应当对错误群集的程序段进行重点测试，以提高测试投资的效益。

在被测程序段中，若发现错误数目多，则残存错误数目也比较多。这种错误群集性现象，已为许多程序的测试实践所证实。例如美国 IBM 公司的 OS/370 操作系统中，47%的错误仅与该系统 4%的程序模块有关。

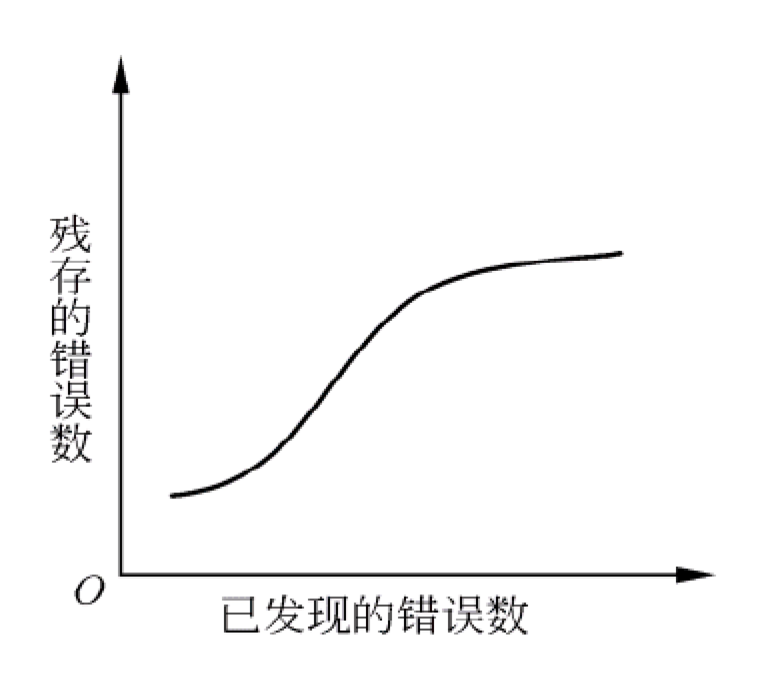



图 2-7　错误的群集现象

> 技巧：一旦测试人员发现在某个模块的 Bug 有集中出现的现象，则应该对这些缺陷集中的模块进行更多的测试和回归验证。

**5. 同化效应**

一名测试人员在同一个项目待的时间越久，越可能忽略一些明显的问题。例如对于界面操作，由于测试人员重复使用同一个软件而产生熟练感，因此可能会忽视一些易用性问题和用户体验问题。

同化效应主要体现在以下两方面：

- 测试人员与开发人员一起在某个项目中工作较长一段时间后，容易受到开发人员对软件观点的影响，变得容易赞同开发人员的观点。
- 测试人员对软件的熟悉程度越高，越容易忽略一些看起来较小的问题。这也是一些测试人员感觉越来越难发现 Bug 的原因。

同化效应会造成 Bug 的“免疫”效果，因此，在测试过程中需要通过轮换，或补充新的测试人员来避免同化效应。

> 技巧：交叉测试能避免一些测试的“盲点”，充分采纳不同人员对软件的不同观点。通过引入新的测试思维来打破测试的局限性。

## 2.5 软件测试的生命周期和过程模型

接下来我们学习软件测试的生命周期。图 2-8 展示了软件测试的生命周期。

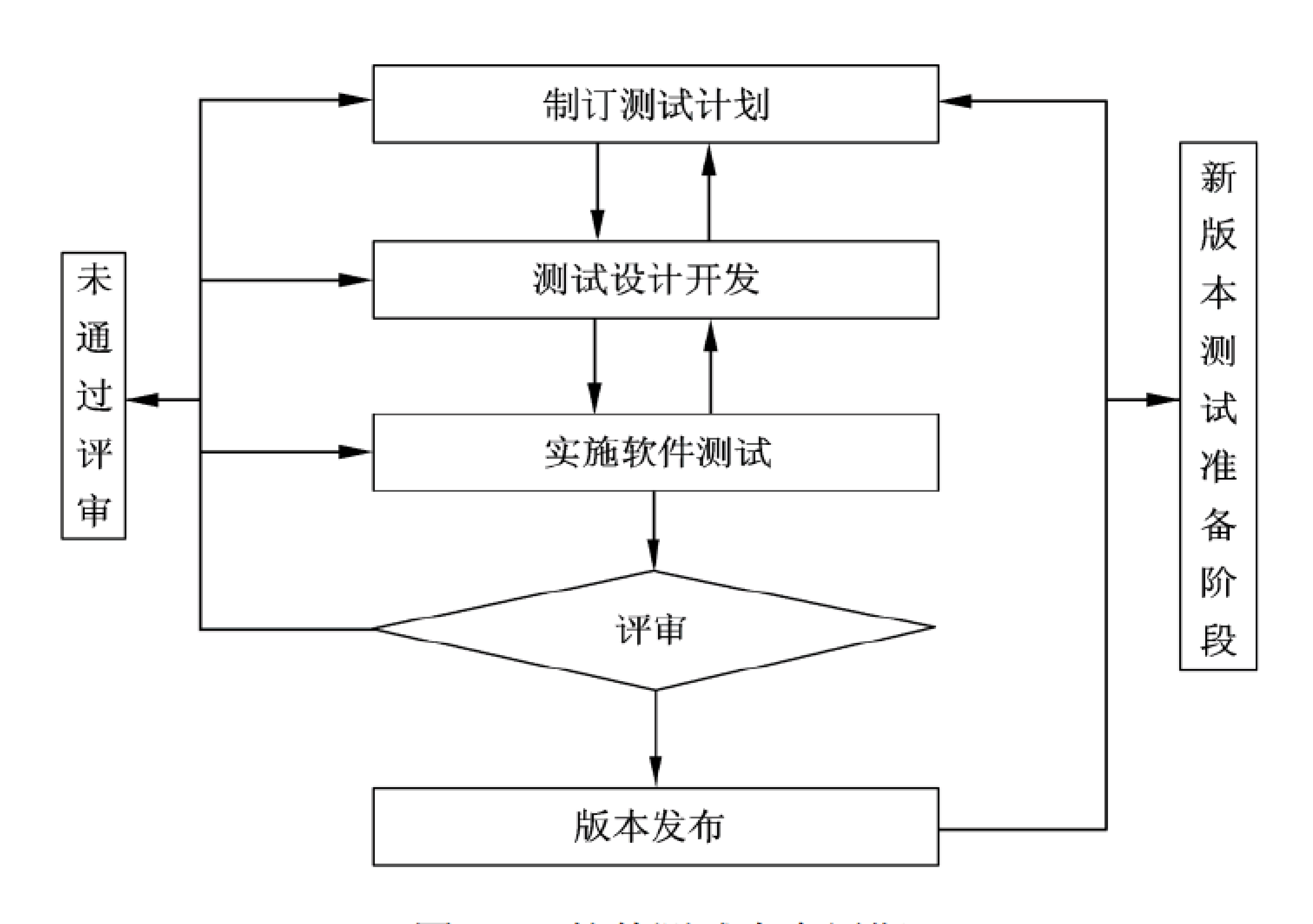



图 2-8 软件测试生命周期

实践证明，尽管人们在开发软件的过程中使用了许多保证软件质量的方法和技术，但开发出的软件中还会隐藏许多错误和缺陷。规模大、复杂性高的软件更是如此。所以，严格的软件测试对于保证软件质量具有重要作用。

### 2.5.1 工作内容

通过图 2-8 可以看出软件测试活动包括制订测试计划、测试设计和开发、执行测试、评估测试。那么这些活动的具体工作内容包括什么？下面分别加以说明。

制订测试计划阶段是测试的先期准备工作阶段，在该阶段中主要是对将要进行的测试工作做一个整体的规划。测试设计阶段则是参照各种相关文档对测试进行设计的工作，包括测试需求的分析和测试用例的设计两项工作。测试开发阶段则是按照测试设计阶段完成的测试需求分析与测试用例设计的方案进行实施的过程，该过程包括测试用例数据的准备、测试工具的开发、测试脚本的开发等工作。此阶段的工作一直持续到软件测试的结束。

执行测试阶段是将设计与开发两个阶段中设计好的测试方法及测试数据应用于实际的软件测试过程中。最后是测试评估，在测试结束后对整个测试过程与产品进行评估的过程。评估过程包括对测试工作的总结、缺陷数据的分析及测试过程的评估等。

## 2.5.2　软件测试过程模型

在软件开发几十年的实践过程中，人们总结了很多的开发模型，比如瀑布模型、原型模型、螺旋模型、增量模型、渐进模型、快速软件开发（RAD）以及最近比较流行的 Rational 统一过程（RUP）等，这些模型对于软件开发过程具有很好的指导作用，但是，非常遗憾的是，在这些过程方法中，并没有充分强调测试的价值，也没有给测试以足够的重视。软件测试与软件开发紧密相关是一系列有计划系统的活动，软件测试专家通过测试实践总结出了很多很好的测试模型，在这些测试模型中把开发过程进行了很好的总结，体现了测试与开发的融合，下面针对 V 模型作一简单的介绍。

RAD(Rap Application Development)，就是软件开发过程中的一个重要模型，称为快速应用开发模型。其模型构图形似字母 V，所以又称 V 模型。

V 模型是最具有代表意义的测试模型，V 模型最早是由 Paul Rook 在 20 世纪 80 年代后期提出的，V 模型在英国国家计算机中心文献中发布，旨在改进软件开发的效率和效果。

V 模型是软件开发瀑布模型的变种，它反映了测试活动与分析和设计的关系，从左到右，描述了基本的开发过程和测试行为，明确地标明了测试工程中存在的不同级别，清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系。

如图 2-9 所示，V 模型大体可以划分为下面几个不同的阶段步骤，即需求分析、概要设计、详细设计、编码、单元测试、集成测试、系统测试和验收测试。

V 模型中，单元测试是基于代码的测试，最初由开发人员执行，以验证其可执行程序代码的各个部分是否已达到了预期的功能要求；集成测试验证了多个单元之间的集成是否正确，并有针对性地对详细设计中所定义的各单元之间的接口进行检查；在所有单元测试和集成测试完成后，系统测试开始模拟系统的运行，以验证系统是否达到了在概要设计中所定义的功能和性能；系统测试应检测系统功能、性能的质量特性是否达到系统要求的指标；验收测试确定软件的实现是否满足用户需要或合同的要求。

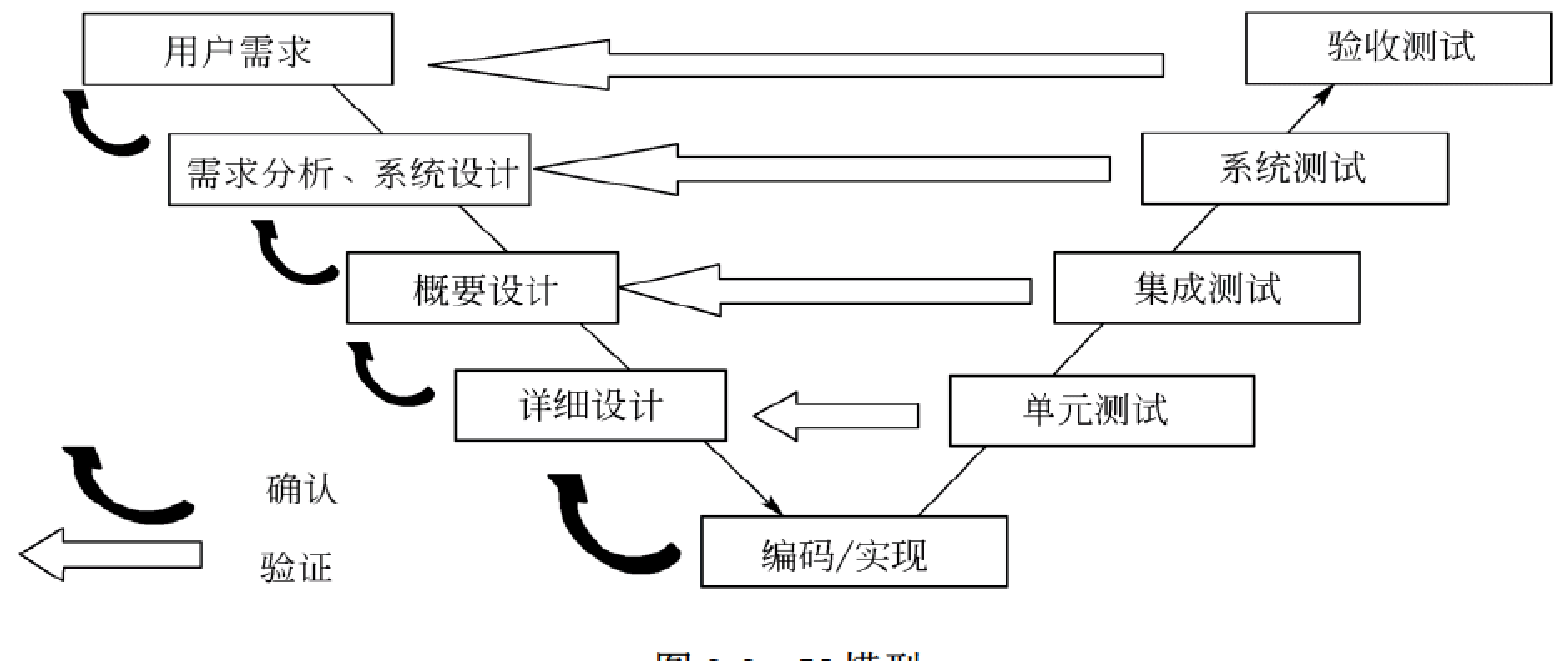



图 2-9　V 模型

但是V模型也存在一定的局限性，它仅仅把测试作为在编码之后的一个阶段，测试是针对程序运行的寻找错误的活动，在需求分析、系统设计阶段引入的错误需要到系统测试和验收测试的时候才能发现。

### 2.5.3 测试模型的使用

在上节中，我们介绍了典型的测试模型V模型，应该说这些模型对指导测试工作有重要的指导意义，我们应该尽可能地去应用模型中对项目有实用的价值方面，在实际工作中，我们要灵活运用各种模型的优点，并同时将开发和测试紧密结合，寻找恰当的就绪点，开始测试并反复迭代测试，最终保证按期完成预定目标。

# 第 3 章　软件测试基础理论

**学习目标：**

1．掌握软件测试流程

2．了解软件测试的分类

3．正确认识软件测试

## 3.1　软件项目中的测试流程

这一节我们将测试流程具体到一次软件开发项目中，了解在各个软件开发阶段中，测试人员究竟可以做哪些测试活动。

### 3.1.1　软件测试流程

软件测试虽然是软件生存周期的一个独立阶段，但测试工作却渗透到从分析、设计直到编程的各个阶段中，如测试计划的编写从分析和设计阶段就开始了，而具体的测试工作随编程工作的不断深入也在进行中。在实际工作中，测试环节可分为明显的、同等重要的几个阶段，即需求测试、单元测试、集成测试和系统测试。测试工作中的第六个阶段是验收测试阶段，验收测试无论在规模上或性质上都和系统测试很相似，它们的根本区别在于：前者是内部的，而后者则是受“客户”控制的。图 3-1 是软件测试所经阶段的一般流程。

**1．需求测试**

软件测试 V 模型要求我们在需求阶段就开始制订系统测试的计划，开始考虑系统测试的方法，但这还是不够的。全面的质量管理要求我们在每个阶段都要进行验证和确认的过程。因此在需求阶段我们还需要对需求本身进行测试。这个测试是必要的，因为在许多失败的项目中，70%～85%的返工是由于需求方面的错误所导致的，并且因为需求的缘故而导致大量的返工，造成进度延迟、缺陷的发散，甚至项目的失败，这是一件极其痛苦的事情。因此我们要求在项目的源头（需求）就开始测试。

在接到测试项目的前期，我们需要对被测软件的需求规格说明书、概要设计文档、详细设计文档、数据库设计文档等文档资料进行查阅，重点检查需求规格说明书中是否存在描述不准确、需求定义模糊、测试用例不正确、语言存在二义性等问题。如果不对这些文档进行检查，将可能在后期的测试工作中不断发生系统修改、需求变更等问题。

我们主要从以下几个方面考虑需求测试。

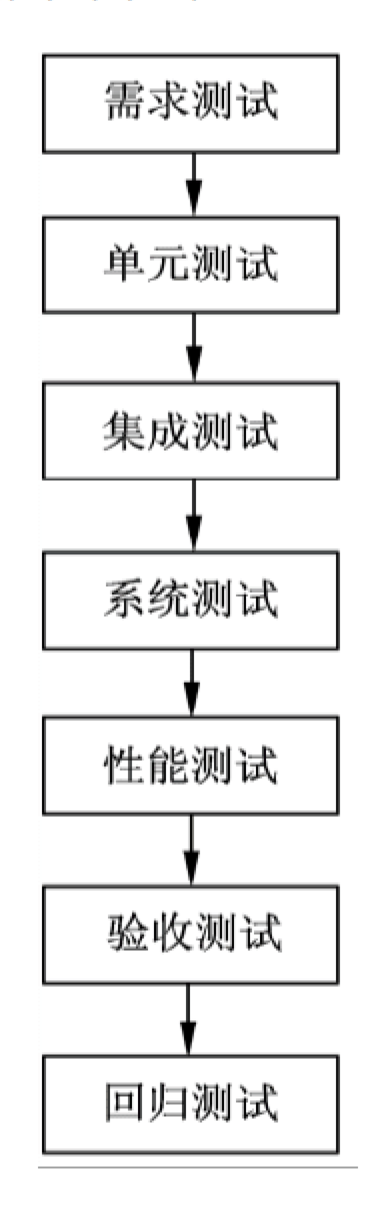



图 3-1 软件测试流程图

（1）完整性。

每一项需求都必须将所要实现的功能描述清楚，从而为开发人员设计和实现这些功能提供所有必要的需求依据。

（2）正确性。

每一项需求都必须准确地陈述其要开发的功能。

（3）一致性。

一致性是指与其他软件需求或高层（系统，业务）需求不相矛盾，或者与我们的项目宣传资料要一致。

（4）可行性。

每一项需求都必须是在已知系统和环境的权能和限制范围内可以实施的。

（5）无二义性。

对所有需求说明的读者都只能有一个明确统一的解释，由于自然语言极易导致二义性，所以尽量把每项需求用简洁明了的用户语言表达出来。

（6）健壮性。

需求的说明中是否对可能出现的异常进行了分析，并且对这些异常进行了容错处理。

（7）必要性。

“必要性”可以理解为每项需求都是用来授权你编写文档的“根源”。要使每项需求都能回溯至某项客户的输入，如需求用例或别的来源。

（8）可测试性。

每项需求都能通过设计测试用例或其他的验证方法来进行测试。

（9）可修改性。

每项需求只应在 SRS（软件需求规格说明书）中出现一次。这样更改时易于保持一致性。另外，使用目录表、索引和相互参照列表方法将使软件需求规格说明书更容易修改。

**2. 单元测试**

单元测试又称为模块测试，顾名思义，就是对程序代码中最小的设计模块单元进行测试。单元测试是在软件开发过程中要进行的最低级别的测试活动。在单元测试活动中，我们主要采用静态测试与动态测试相结合的办法。首先采用静态的代码走查，检查程序代码中不符合编程规范，存在错误或者遗漏的地方，同时使用代码审查的方法，小组检查项目代码，以期发现更多的问题，然后再使用单元测试工具，比如 JUnit 等工具进行程序代码内逻辑结构、函数调用等方面的测试。据业界统计，单元测试一般可以发现大约 80%的软件缺陷。

单元测试对测试人员的要求相对较高，一般需要有几年的代码编写经验，并且要十分熟悉当前的被测系统，以及该系统是否与其他系统的接口关联情况。在大多数公司中，单元测试一般情况下由对应的开发工程师负责。

在实际的工作过程中，很多人认为单元测试没有必要，认为这个过程耽误了工作进度，没什么实际的效果，其实不然，软件测试活动贯穿于软件生产的整个过程，每个环节的检查测试都是不可缺少的，我们知道缺陷发现得越早，其修复的代价也就越小。单元测试在编码阶段占据着非常重要的地位。编码只是一方面，还需检查编码，保证代码的质量，所以说，在软件生产过程中及时地开展单元测试是非常有必要的，可以降低编码的错误率，提高编码质量。

**3. 集成测试**

我们知道，个体不代表全部，在单元测试阶段，发现并解决了部分的问题，但不能解决所有的问题。在当前环境下，我们可能觉得代码质量已经非常好了，至少我们的能力范围内不能再发现缺陷了。此时，仅仅是一个代码模块、功能模块，一旦组合起来，可能相互之间的问题就会暴露了。

集成测试，又称为组装测试，就是将软件产品中各个模块组装起来，检查其接口是否存在问题，以及组装后的整体功能、性能表现。在开展集成测试之前，我们进行了深入的单元测试（当然，实际工作中大多数公司不会做单元测试，仅有程序员各自检查自己的代码），从个体来讲，可能解决了很多的缺陷，但所有的个体组合起来，就可能出现各种各样的问题。1+1<2 的问题，此刻尤为突出。

在单元测试阶段，我们无法发现资源争用、接口调用、时钟延迟等问题。假如有两个模块 A，B，一个数据在 A 模块处理时因代码问题，延迟了 0.01 s，然后流转到 B 模块处理，又延迟了 0.01 s，那么对于个体来讲，可能 0.01 s 算不了什么，但当数据流转

的环节增加时，相应的延迟时间也在不断地增加，最终的累加数据可能带来非常严重的后果。每个环节的缺陷在最终被放大后，可能会引起软件的失效。所以，单元测试阶段的成果并不能保证集成测试没有问题。采用科学有效的集成测试方法，在软件生产活动中是非常必要的。

**4. 系统测试**

系统测试是将通过集成测试的软件部署到某种较为复杂的计算机用户环境进行测试，这里所说的复杂的计算机用户环境，其实就是一般用户的计算机环境。比如，我们开发了一套财务软件，软件中采用了 Excel 2000 版中的某些绘图控件，在集成测试阶段，我们大多数是在一种比较干净的系统进行测试。所谓的干净，就是在测试机上没有多余的软件，仅有所需的操作系统和被测软件。当集成测试完成后，就将被测软件置入比较复杂的运行环境中，进行集成测试。在这个过程中，我们往往有很大的收获，比如进行安装测试的时候，会发现在集成阶段安装没有问题，而在复杂的用户环境下，却不能安装。举个例子，前面所说的财务软件，当我们的计算机中装有新版本的 Excel 时（如 Excel 2003），就可能出现无法正常使用该软件的问题，因为该财务软件所需的控件在 Excel 2003 可能不存在，或者不适用。类似于这样的问题，就需要我们将被测软件置入一个较为普遍的用户计算机环境中。

系统测试的目的在于通过与系统的需求定义作比较，发现软件与系统的定义不符合或与之矛盾的地方。这个阶段主要进行的是安装卸载测试、兼容性测试、功能确认测试、安全性测试等。系统测试阶段采用黑盒测试方法，主要考察被测软件的功能与性能表现。如果软件可以按照用户合理期望的方式来工作的时候，即可认为通过系统测试。

系统测试过程其实也是一种配置检查过程，检查在软件生产过程中是否有遗漏的地方，在此时做到查漏补缺，以确保交付的产品符合用户质量要求。

**5. 性能测试**

软件测试工作对于一般的软件产品而言，主要测试 4 个方面：文档、界面、功能和性能。这里，我们来了解一下性能测试的概念。

性能测试所包含的内容比较多，按照典型的概念来理解，就是要求被测软件在业务处理速度、处理能力和所耗用的硬件系统资源比率满足用户的需求。举个例子，对于某个论坛，我们需要测试该论坛支持 10 000 个用户的同时使用，并且在这种情况下，打开帖子的速度能否控制在 4 秒钟以下，论坛服务器的 CPU 使用率不超过 80%，内存占用率不超过 75%等，这些就是典型的性能测试指标。实施性能测试，一方面可以验证被测软件是否符合用户需求；另一方面，可以得到相关的性能数据，为被测软件的优化提供参考。

随着 B/S 结构软件流行，越来越多的用户希望他们的软件能够有比较好的性能表现，于是性能测试工作的地位被提升，重要性也日渐显露。那么具体如何实施性能测试呢?

我们知道，计算机的处理速度是非常快的，如果想考察某项功能是否达到我们的需

求，比如业务处理速度，我们不可能手动记录处理时间，因为这样是不现实的，也是不明智的。同时，有些时候我们需要模拟大量用户使用系统的情况，比如测试新浪网能否承受 100 万网络用户的并发访问，那么是不是要找 100 万个人、100 万台电脑在同一时刻登录新浪网呢?显然不现实，也不太可能。一系列实际的问题告诉我们，不要尝试用手动方式进行性能测试，应当编写一段相应的程序或者使用专门的工具进行，比如利用 LoadRunner 自动化性能测试工具。

性能测试的难度相对来说比较大，要求测试人员掌握编程语言，精通业务流程，拥有深厚的项目经验。所以，想顺利地开展性能测试，需要测试工程师不断学习，掌握相应的知识。

**6. 验收测试**

在系统测试完成后，将会进行用户测试。这里的用户测试，其实可以称为用户确认测试。在正式验收前，需要用户对本系统做出一个评价，用户可对交付的系统做测试，并将测试结果反馈回来，进行修改、分析。面向应用的项目，在交付用户正式使用之前要经过一定时间的用户测试。

用户测试在整个软件生产流程中非常重要，这个环节是被测软件首次作为正式的系统交由用户使用，用户会根据他们的实际使用情况进行测试、试用，并提出实际使用过程中的问题。我们知道，软件测试是尽可能地去模拟客户的业务行为，遵循既定的用户需求和软件生产规范，寻找软件产品中的缺陷。然而，测试工程师并不是真正的最终用户，所以，在测试过程中仍旧会存在一些未能发现的实际业务缺陷，这对软件质量的保证并不是一个好消息。所以，在产品正式发布前，加入用户的测试是一个明智的选择，因为用户能从最终的业务角度来试用系统，并能发现很多有价值的缺陷，从某个角度来说，用户测试是软件生产流程中的最后质检关。

**7. 回归测试**

简单说，回归测试就是过一段时间以后再回过头来对以前修复过的 Bug 重新进行测试，看该 Bug 是否会重新出现。

回归测试一般发生的情况在用户发现缺陷反馈到公司后，测试部门将组织一次回归测试。或者，公司内部人员发现某软件存在某种缺陷的时候，也可以发起回归测试。回归测试阶段主要的目的是检查以前的测试用例能否再次通过，是否还有需要补充的用例等。

有些公司会采用自动化测试工具来进行回归测试，比如利用工具，对于产品级，变动量小的软件而言，我们可以利用工具去执行测试。但一般情况下，都由测试工程师手动地执行以前的测试用例，来检查用例通过情况。

回归测试可以发现在产品发布前未能发现的问题，比如时钟的延迟问题、软件的性能问题等。

### 3.1.2 需求分析阶段的测试活动

通过第 2 章中提到的软件生命周期模型可以知道，软件项目在完成了前期的可行性分析和项目计划后，就进入了需求分析阶段，事实上一个软件项目或产品的成败与需求分析有着非常重要的联系。因此在没有明确用户需求的情况下盲目地进行开发和测试都不能够取得理想的效果。

根据所提到的软件测试的概念：验证软件是否符合用户需求。软件开发活动进入到需求分析阶段，自然就涉及了用户的需求，那么测试人员就有必要开始他们的工作，来发现软件开发需求与用户实际需求之间的差异了。

若具备条件，测试人员应从客户需求调研阶段就介入到项目中。软件产品需求阶段工作流程如图 3-2 所示。

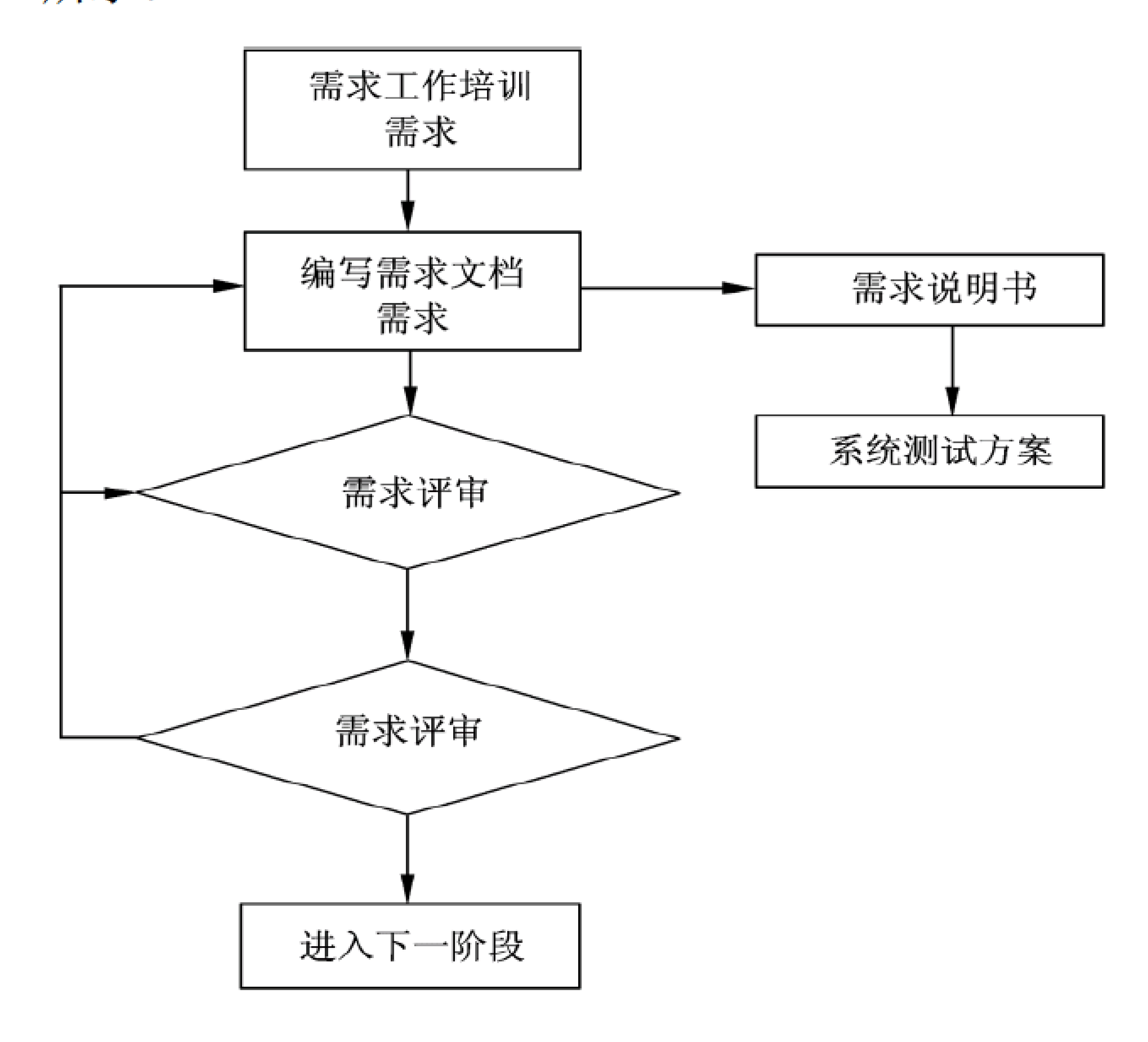



图 3-2 软件产品需求调研阶段工作流程

通过软件产品需求调研阶段工作流程图可以看到，在这一阶段有两个与软件测试相关的输出，它们分别是对软件需求规格说明书的测试和编写系统测试方案。

对需求规格说明书的测试也是软件的早期测试内容之一，主要是通过测试人员的测试工作，发现用户的实际需求与需求分析人员制定的需求规格说明书的差异，以避免由于需求分析的错误而导致的后期软件修改成本的增加。

另外，项目进入需求分析阶段后，由于将来系统的各项需求已经明确，所以在此阶段，测试工程师完全可以参照需求规格说明书，将此软件的系统测试方案制作完成。

### 3.1.3　软件设计阶段的测试活动

需求调研阶段完成后，人们会根据需求说明书的要求开始设计软件，包括概要设计阶段和详细设计阶段，然后开发人员根据产品的详细设计进行编码，这一过程叫做软件设计和编码阶段。软件设计和编码阶段的工作流程如图 3-3 所示。

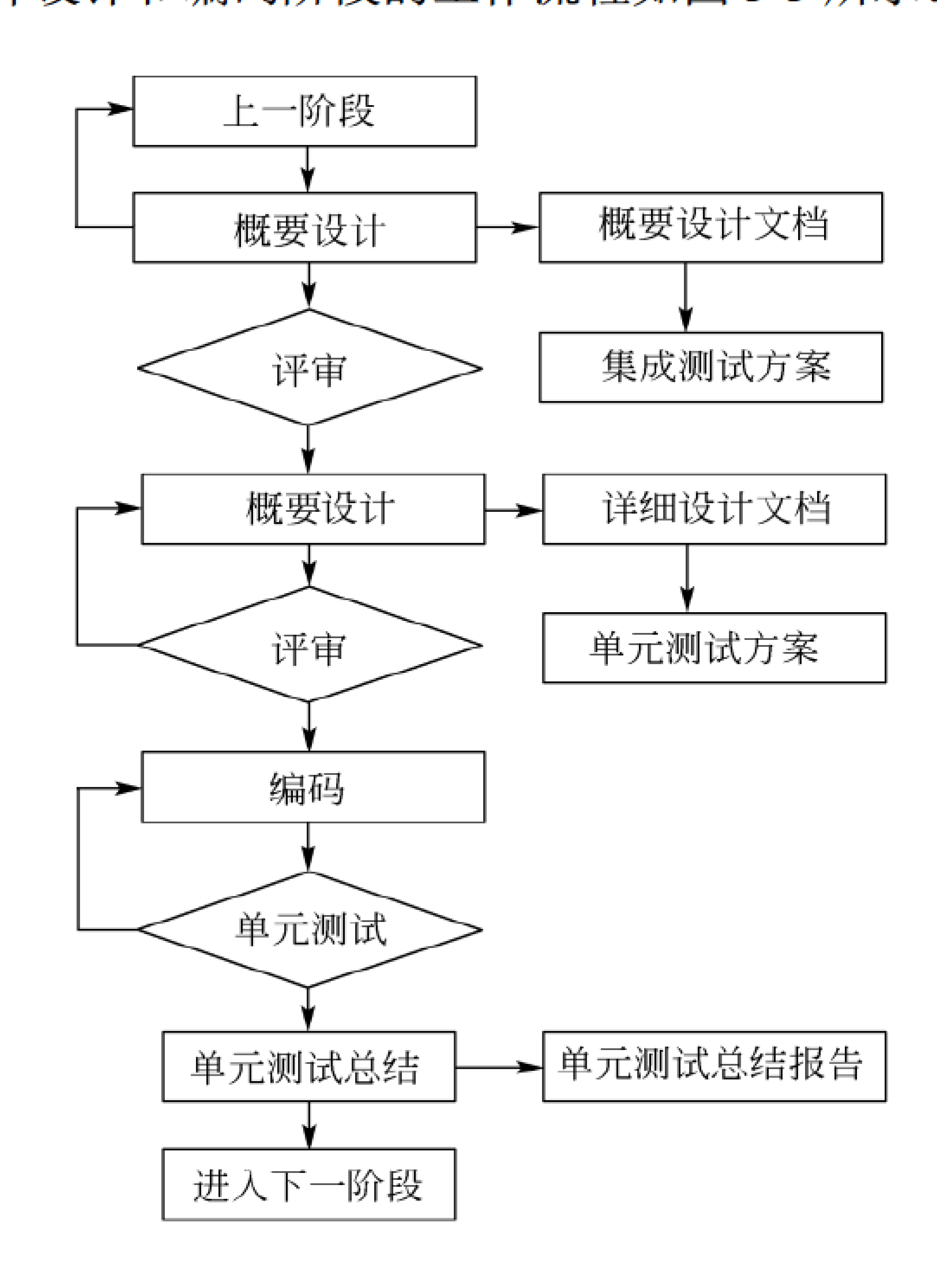



图 3-3　软件设计和编码阶段工作流程

那么根据图 3-3，我们可以看到，在软件设计和编码阶段的测试活动有以下几个方面的内容。首先，根据设计阶段生成的概要设计编写集成测试方案；然后，在软件开发流程进入详细设计阶段后，根据生成的软件详细设计报告生成单元测试方案。

当开发工程师根据软件设计人员的详细设计，开始软件编码过程后，测试人员就可以进入单元测试流程了，单元测试完成后生成单元测试总结报告。

### 3.1.4　其他测试活动

软件在完成了各个单元的测试过程之后，就会将各个模块、单元拼接成该软件的子系统，直到最后会拼装成为一个完整的软件系统。那么，在这些拼装软件的过程中，测试阶段就进入到集成、系统、验收测试阶段，该阶段的工作流程如图 3-4 所示。

通过以上的分析，可以得出这样一个结论：软件测试工作贯穿了整个软件生命周期，渗透到软件开发需求、设计、实现的各个阶段中，如系统测试方案的编写从需求分析阶段就开始了，而具体的测试工作随编程工作的不断深入也在进行中。

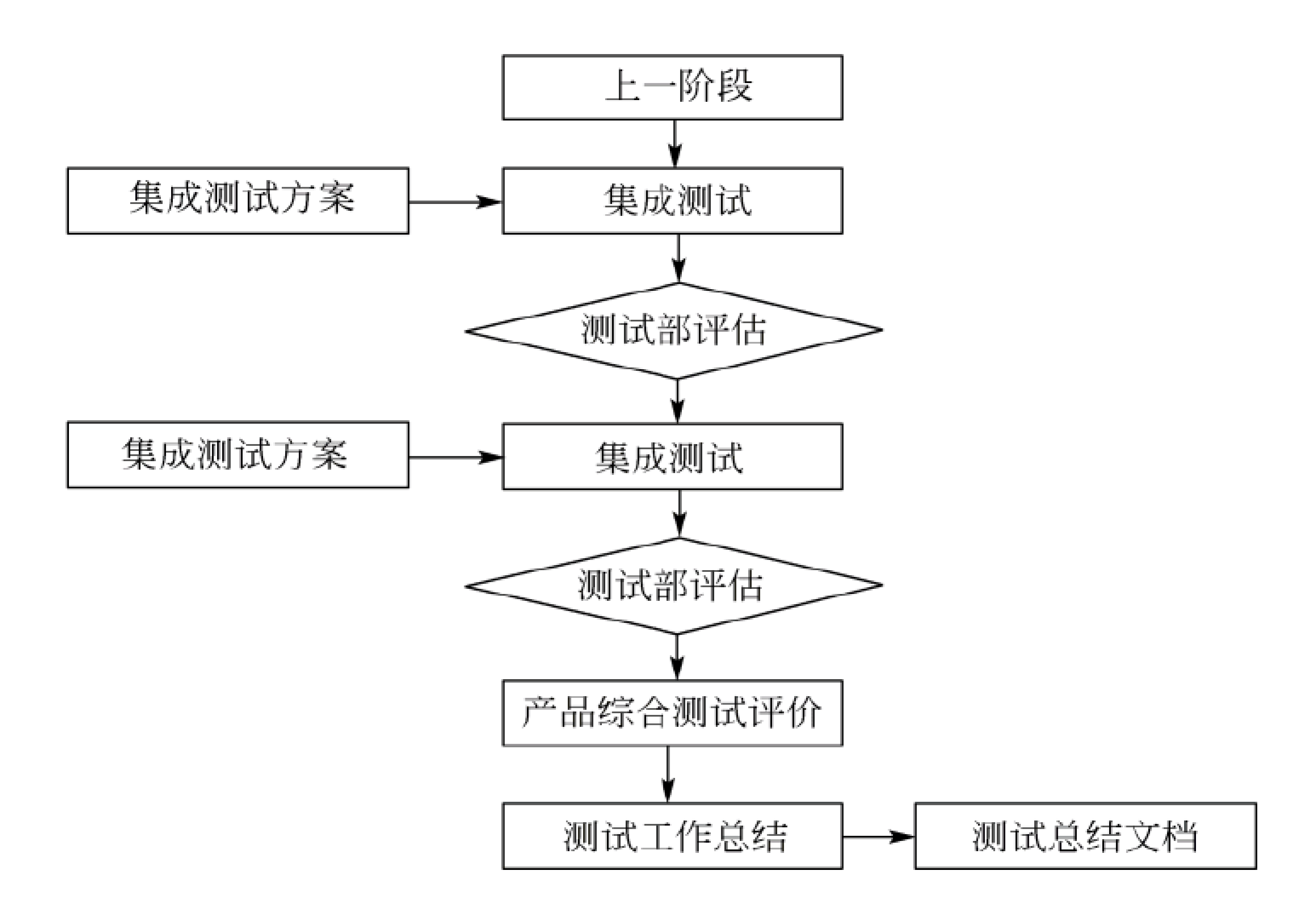



图 3-4　集成、系统、验收测试阶段的测试工作流程

## 3.2　软件测试的基本分类

在了解了软件测试定义后，我们来看看软件测试都有哪些分类。一般地，我们将软件测试活动分为以下几类：黑盒测试、白盒测试、灰盒测试、静态测试、动态测试、手动测试、自动化测试等。

**1．黑盒测试**

软件测试行业，最常听到的名词就是黑盒测试，那么到底什么是黑盒测试呢?

黑盒测试又叫功能测试、数据驱动测试或基于需求规格说明书的功能测试。该测试类别注重于测试软件的功能性需求。

采用这种测试方法，测试工程师把测试对象看作一个黑盒子，完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明。如图 3-5 所示，测试工程师无须了解程序代码的内部构造，完全模拟软件产品的最终用户使用该软件，检查软件产品是否达到了用户的需求。举个例子，我们购买了手机以后，很少有人拆开手机观察其内部的结构，大多数情况下，我们只是使用该手机的功能，从某种意义上说，此时这部手机就是我们的测试对象，所采用的测试方法就是黑盒测试。




图 3-5　黑盒测试示例图

黑盒测试方法能更好更真实地从用户角度来考察被测系统的功能性需求实现情况。在软件测试的各个阶段，如单元测试、集成测试、系统测试等阶段中都发挥着重要作用，尤其在系统测试中，其作用是其他测试方法无法取代的。

**2．白盒测试**

与黑盒测试相对的软件测试方法，称为白盒测试。白盒测试又称结构测试、逻辑驱动测试或基于程序代码内部构成的测试。此时，测试工程师将深入考察程序代码的内部结构、逻辑设计等。就像前面手机的例子，我们拆开手机，观察手机电路板的设计，液晶屏的构成等。白盒测试需要测试工程师具备很深的软件开发功底，精通相应的开发语言，一般的软件测试人员难以胜任该工作。图 3-6 是白盒测试的示例图，对于白盒测试工程师来说软件产品的内部构成是敞开的。

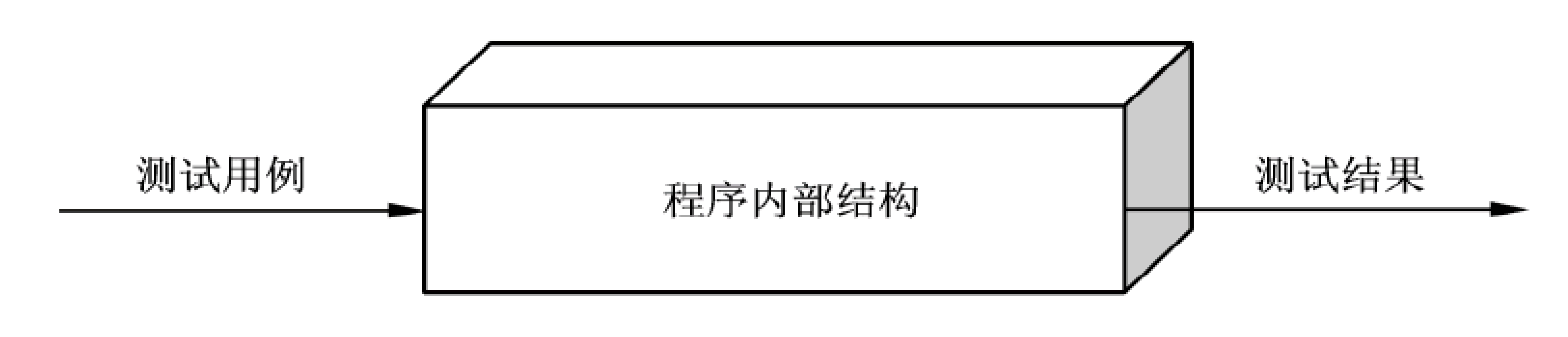



图 3-6　白盒测试示例图

**3．灰盒测试**

与前面的黑盒测试、白盒测试相比，灰盒测试介于两者之间。黑盒测试仅关注程序代码的功能性表现，不关注内部的逻辑设计、构成情况，白盒测试则仅从程序代码的内部构成考虑，检查其内部代码设计结构、方法调用等。而灰盒测试结合这两种测试方法，一方面考虑程序代码的功能性表现；另一方面，又需要考虑程序代码的内部结构。通俗地讲，灰盒测试就是白加黑，像我们的性能测试、自动化功能测试就采用了灰盒测试的方法，如图 3-7 所示。

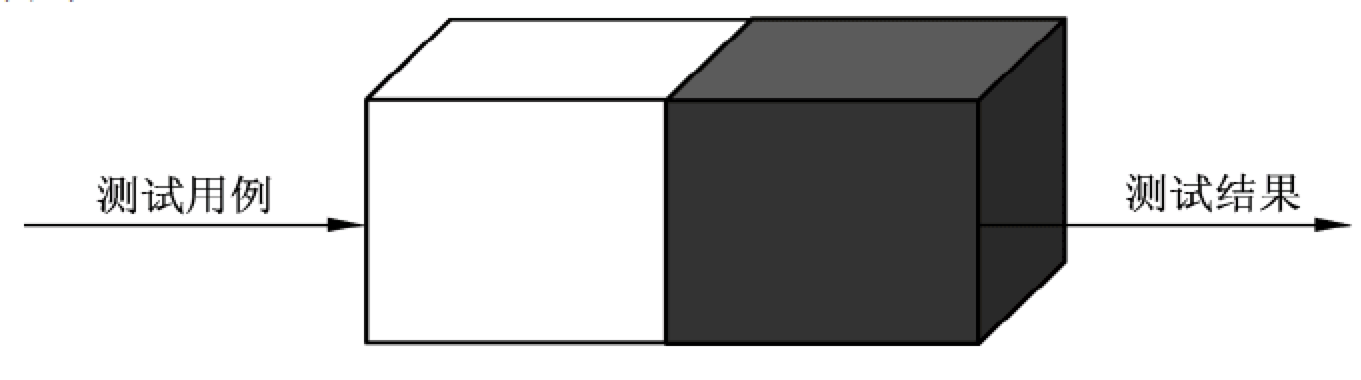



图 3-7　灰盒测试示例图

**4. 静态测试**

静态测试，顾名思义，就是静态的、不执行被测对象程序代码而寻找缺陷的过程。通俗地讲，静态测试就是用眼睛看，阅读程序代码、文档资料等，与需求规格说明书中的客户需求进行比较，找出程序代码中设计不合理以及文档资料有错误的地方。

一般在企业、公司里会召开正规的评审会，通过评审的方式，找出文档资料、程序代码中存在缺陷的地方，并加以修改。

在进行代码的静态测试时，可以采用一些代码走查的工具，如 QA、C++、C++Test 等。

**5. 动态测试**

动态测试即为实际地执行被测对象的程序代码，输入事先设计好的测试用例，检查程序代码运行得到的结果与测试用例中设计的预期结果之间是否有差异，判定实际结果与预期结果是否一致，从而检验程序的正确性、可靠性和有效性，并分析系统运行效率和健壮性等性能状况。

动态测试由 4 部分组成：设计测试用例、执行测试用例、分析比较输出结果、输出测试报告。

动态测试有 3 种主要的方法：黑盒测试、白盒测试和灰盒测试。

**6. 手动测试**

在未真正接触软件测试之前，很多人都认为，软件测试工作就是执行一些鼠标单击的动作来查找缺陷。的确，在手动测试阶段，大部分的测试工作就是模拟用户的业务流程，来使用软件产品，从而发现软件产品中的缺陷。手动测试是最传统的测试方法，也是现在大多数公司都使用的测试形式。它是测试人员设计测试用例并执行测试用例，然后根据实际的结果去和预期的结果相比较并记录测试结果，最终输出测试报告的测试活动。这样的测试方法，可以充分发挥测试工程师的主观能动性，将其智力活动体现于测试工作中，能发现很多的缺陷，但同时这样的测试方法又有一定的局限性与单调枯燥性。

**7. 自动化测试**

随着软件行业的不断发展，软件测试技术也在不断地更新，出现了众多的自动化测试工具，如 HP 的 QuickTest Professional、LoadRunner，微软的 WAS，IBM 的 Rational 等。所谓的自动化测试，就是利用一些测试工具，模拟用户的业务使用流程，让它们自动运行来查找缺陷。也可以编写一些代码，设定特定的测试场景，来自动寻找缺陷。自动化测试的引入，大大地提高了测试的效率和测试的准确性，而且写出的比较好的测试脚本，还可以在软件生命周期的各个阶段重复使用。

自动化测试的优点是能够很快、很广泛地查找缺陷，同时可以做很多重复性的工作，在回归测试阶段，我们可以利用 QuickTest Professional 自动化功能测试工具进行，而无须大量的软件测试人员手动地再次执行测试用例，极大地提高了工作效率。有时候我们常常需要做一种压力测试，需要几万甚至几十万个用户同时访问某个站点，以保证网站

的服务器不会出现死机或崩溃的现象。一般来说，要几万人同时打开一个网站是不现实的，就算能够找到那么多的测试者，成本也很高。但是，利用测试工具，比如 LoadRunner，就可以非常容易地做到，并且测试工具还可以自动判断浏览结果是否正确。

然而，自动化测试的缺点也很明显，它们只能检查一些比较主要的问题，如崩溃、死机，但是却无法发现一般的日常错误，这些错误通过人眼很容易找到，但机器却往往找不到。另外，在自动测试中编写测试脚本工作量也很大，有时候该工作量甚至超过了手动测试的时间。

如今很多的软件产品生产，我们都是追求短（项目周期短）、平（成本低）、快（市场反应快），根本没有那么多的时间留给我们做测试，在这种情况下，如果我们不合时宜地盲目追求自动化测试，可能带来的不是好处，而是项目的失败。

需要注意的是，在自动化测试活动中，测试工具的应用可以提高测试的质量、测试的效率。但是在选择和使用测试工具的时候，我们也应该看到，在测试过程中，并不是所有的测试工具都适合我们使用，同时，有了测试工具、会使用测试工具并不等于测试工具真正能在测试中发挥作用。因此，我们应该根据实际情况选择或者不选择测试工具，选择使用何种测试工具，千万不能为了使用工具而刻意地去使用工具。

## 3.3　正确认识软件测试

### 3.3.1　软件测试与建立软件信心的关系

软件测试是对软件建立信心的一个过程。测试是评估软件或系统品质或能力的一种积极的行为，是对软件质量的一种度量。软件信心与软件测试的关系可用图 3-8 来描述。对软件进行的测试越充分，人们对这个软件的信心越强。可以想象，当将软件提交给用户使用时，告诉用户软件没有被测试，用户的表情将会是什么样子。

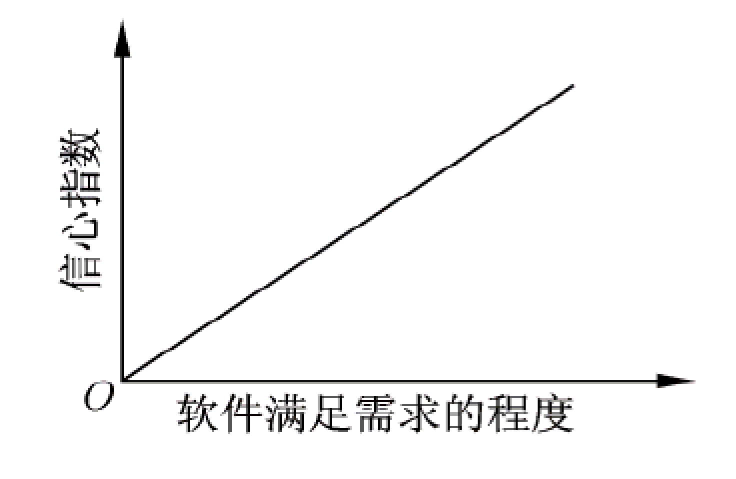



图 3-8　软件信心建立在软件对需求程度的度量上

### 3.3.2　软件测试的两面性

从软件测试的目的出发，可以把软件测试分为两类：

一类是为了验证程序能正常工作的测试。另外一类是为了验证程序不能正常工作的测试。

一类测试是正向进行的，另一类测试是反向进行的，测试人员应该从两面“夹击”，如图 3-9 所示。

验证程序能正常工作需要有一定的依据，软件需求文档就是一种好的依据。但是，如果需求文档本身就是错误的呢？因此，不能只依靠需求文档来验证程序是否能正常工作，还需要有经验的测试人员的判断和对软件的理解。

要验证程序在所有情况下都能正常工作不是一件容易的事情，甚至可以说这是不可能完成的任务。因为，现在的软件程序越来越复杂，程序的状态空间变得越来越广，在有限的测试时间和测试资源下，要想证明程序在所有情况下都能正常工作是不可能的。

相比之下，验证程序不能正常工作会相对容易一些，只要能找到错误，就能证明软件是不正确的。但是，要想找到所有的错误也不是一件容易的事，因为 Bug 会随着程序的修改变得越来越少，变得越来越隐蔽，越到后面越难发现错误，如图 3-10 所示。

目前，大部分软件测试组织均综合采用上述两种测试方式。主要体现在以下方面：

- 用例的设计分正方向和反方向的测试，及主成功场景用例和扩展场景用例的测试。
- 将严格的测试用例执行过程及灵活的探索性测试执行过程相结合。
- 软件测试的中前期主要集中精力发现软件的错误，中后期主要集中精力于验证软件的正确性。
- 单元测试主要关注程序做了正确的事情，集成测试和系统测试主要关注程序的错误行为。
- 自动化测试主要专注于验证程序的正确行为，手工测试主要专注于发现软件的错误行为。

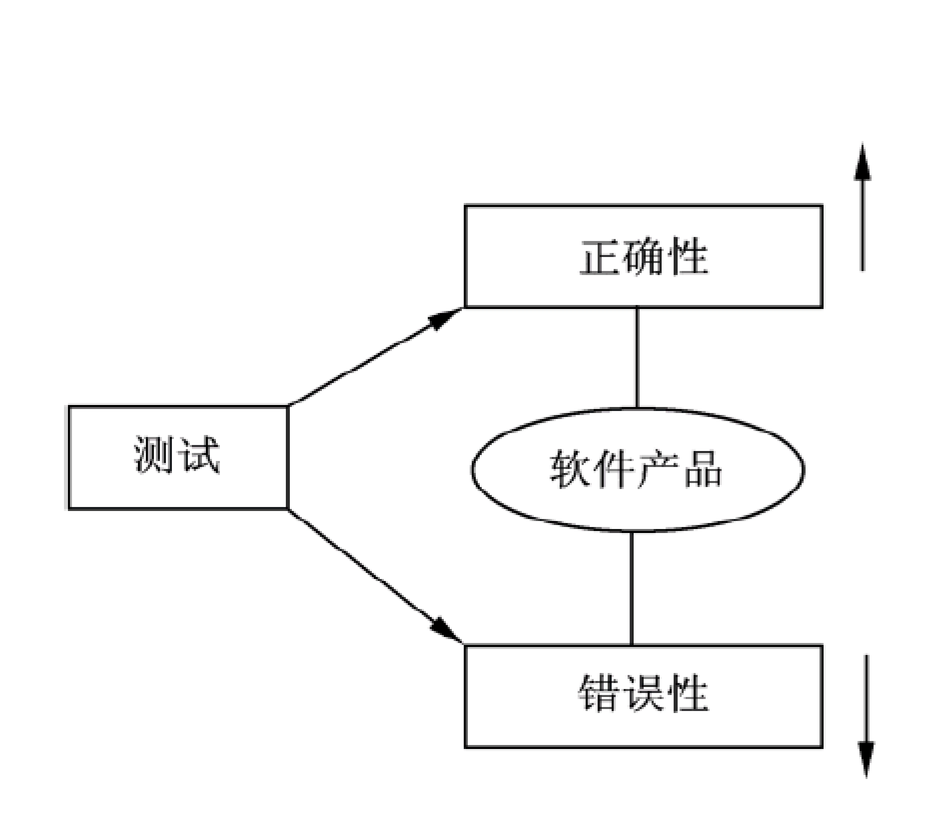



图 3-9　软件测试的两面性

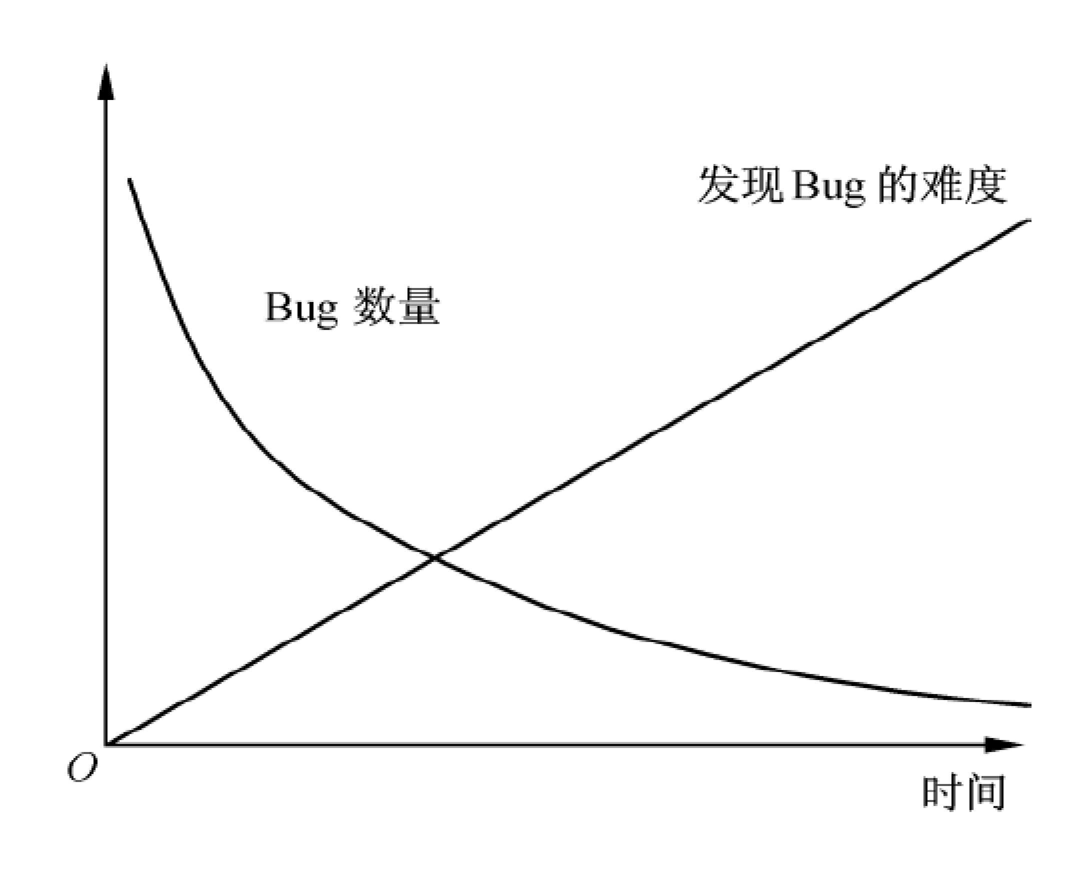



图 3-10　Bug 发现率

### 3.3.3　测试是一种服务

软件测试是一种服务，软件测试人员对软件产品进行研究和探索，获取有关软件的各种信息，供项目决策者做出正确的决定，如图 3-11 所示。

将软件测试理解成一种服务的好处主要有以下两方面：

- 测试被理解成服务可能会让某些测试人员感到失落，但是这种理解的方式却可以化解很多测试人员与开发人员之间的矛盾。
- 有利于客观公正地进行测试工作。

这种对测试的理解综合了对软件测试目的的两种观点。

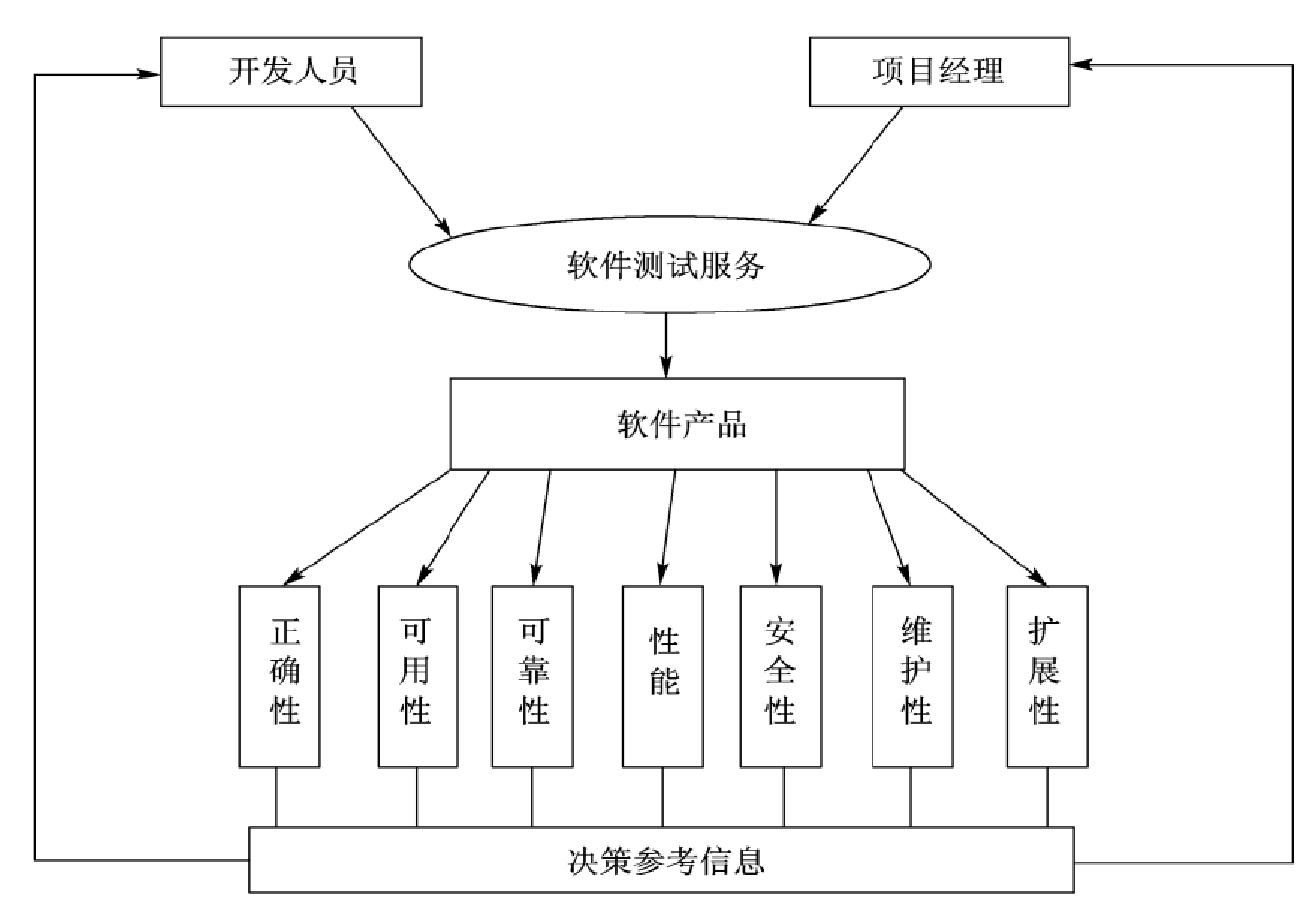



图 3-11　测试作为一种服务被调用

# 第 4 章　软件测试项目与组织

**学习目标：**

1. 了解软件测试的工作流程
2. 了解测试部门的组织结构

## 4.1　软件测试的工作流程

知道软件测试的相关概念后，对于一个企业或者公司的软件测试工作流程是什么样子的呢?没有工作经验或者不太了解软件测试工作的朋友可能没有一个清晰的概念，本章将从实际角度出发，介绍一般软件公司中软件测试部门的工作开展流程。在这之前，我们先来看看测试部门的组织结构情况。

### 4.1.1　测试部门组织结构

从不同的角度出发，测试部门的构成可从这几个方面考虑：人员构成、技术构成、资源构成。实际上，一个测试部门的管理者，可从这些方面考虑部门的组织结构。

**1．人员构成**

一个完整的测试部门，一般包括以下几个角色：测试主管、测试组长、环境保障人员、配置管理员、测试设计人员和测试工程师。如图 4-1 所示。

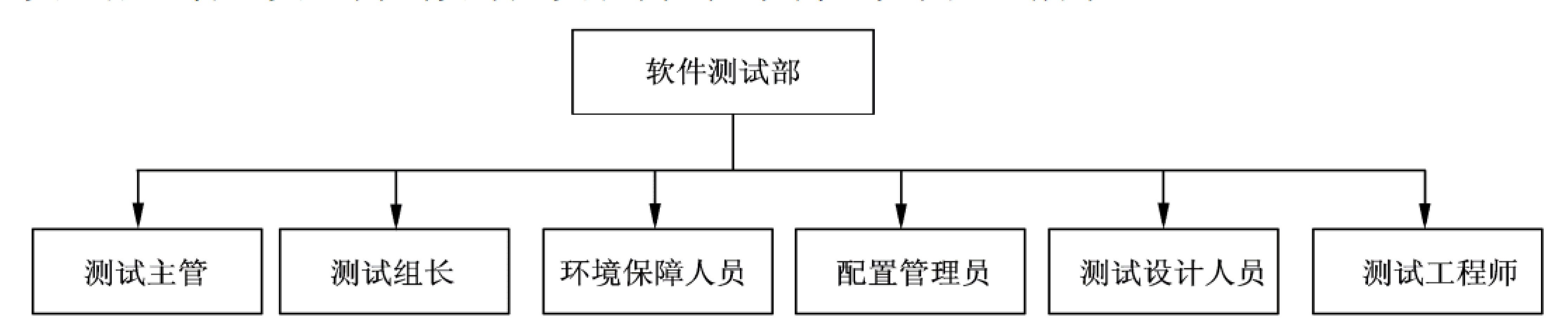



图 4-1　测试部门组织结构图

（1）测试主管。

测试主管负责测试部门的日常管理工作，负责部门的技术发展、工作规划等，同时他也是测试部门与其他部门的接口人，在其他兄弟部门需要测试部门协助或安排测试工作的时候，需要首先与测试主管沟通，提出申请。

（2）测试组长。

测试组长隶属于测试部门，由测试主管指派。在接收到一个项目测试需求后，测试主管会根据项目的实际情况，比如项目的技术要求、难易程度，指派合适的测试人员担当测试组长角色，由其负责该项目测试工作。有些公司称测试组长为测试经理。

（3）环境保障人员。

环境保障人员的作用是维护整个项目过程中的系统环境，如硬件、软件方面的。一般的公司不具备这样的人员，都由测试人员兼做，也可能有专职的保障人员，但不隶属于测试部门。所以该角色一般是重叠的。

（4）配置管理员。

配置管理是软件开发过程中一个极其重要的工作流程，在这个环境可以对需求变更、版本迭代、文档审核起到相当大的作用，所以稍微正规一些的公司都会配备配置管理员。

（5）测试设计人员。

一般由高级测试工程师担当，负责项目测试方法设计、测试用例设计以及功能测试、性能测试的步骤、流程设计。很多公司将该角色与测试工程师重叠，不严格区分测试设计人员与测试工程师角色。

（6）测试工程师。

测试工程师的实际工作内容大多数是执行测试用例，进行系统的功能测试，经过多次的版本迭代，完成系统测试。一般由初级测试工程师、中级测试工程师担当。

**2．技术构成**

技术构成主要是从测试部门需具备的技术角度来考虑，主要有这几类：白盒测试技术人员、黑盒测试技术人员、自动化测试技术人员、项目管理技术人员等，如图 4-2 所示。

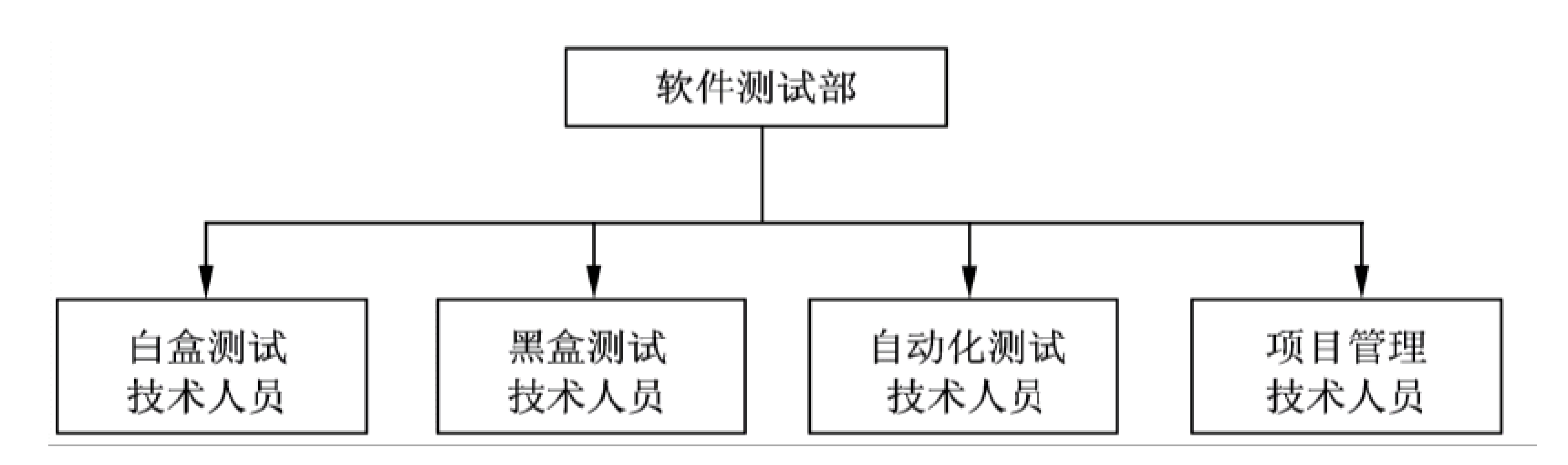



图 4-2　测试部门技术构成图

（1）白盒测试技术人员。

该职位需要测试人员精通掌握软件的开发语言，一般需要有几年的开发经验，能够进行底层的代码 review、测试桩设计等，同时能够使用白盒测试工具对系统的最小功能单元进行测试，找出代码、系统架构方面的缺陷。

（2）黑盒测试技术人员。

黑盒测试技术人员一般要求测试人员具有一定的软件工程理论、软件质量保证知识，需要从系统的功能实现、需求满足情况监察系统的质量，需要掌握基本的软件开发语言、数据库基本知识、操作系统基本知识、测试流程以及相应的工作经验。

（3）自动化测试技术人员。

自动化测试技术人员相对的要求较高，需要测试人员掌握软件开发的知识，系统的调优，自动化测试工具，如 QuickTest Professional、LoadRunner 等，同时需要具备相当丰富的工作经验。目前国内这方面的人才比较紧缺。

（4）项目管理技术人员。

该角色要求项目管理人员掌握一般常用的项目管理知识，如配置管理、版本控制、评审管理、项目实施与进度控制等，不一定具备多强的测试技术，但需要有丰富的项目管理经验以及沟通协调能力，能够保证项目在一个可控的环境下稳定运作。

**3．资源构成**

资源构成主要考虑的是测试部门的组建需要哪些硬件、软件资源，主要包括：硬件资源、软件资源、技术支持等，如图 4-3 所示。

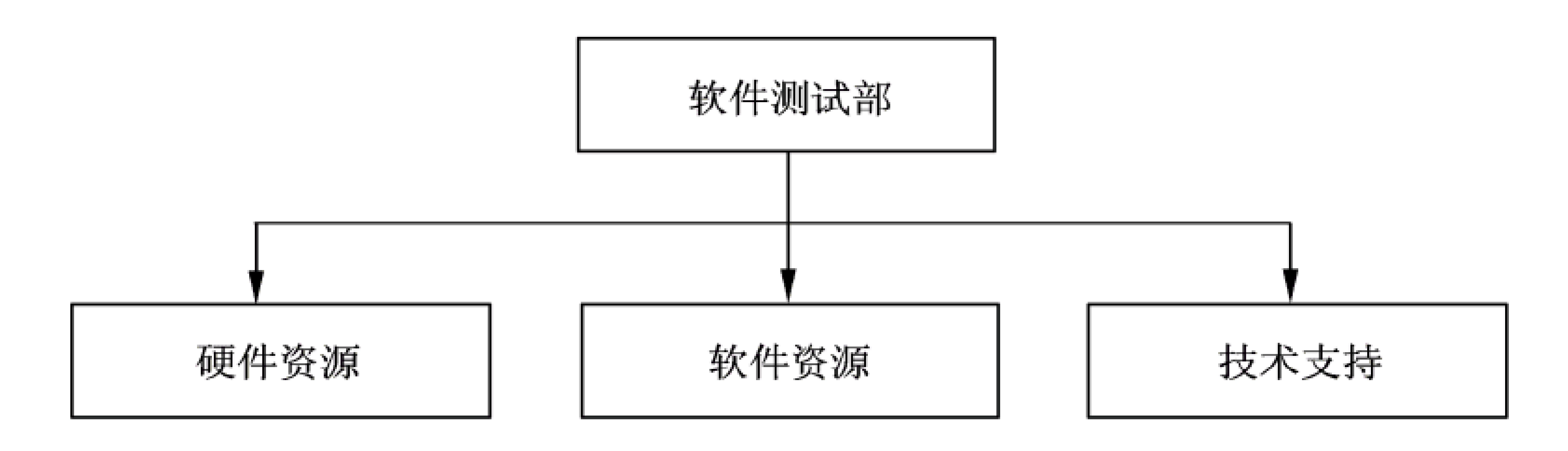



图 4-3 测试部门资源构成图

（1）硬件资源。

测试部门面临的是复杂多变的用户环境，需要在不同环境下进行系统的测试，所以就需要有齐备的测试环境，比如测试 PC、测试服务器、测试芯片、测试手机等，需要模拟各式各样的用户环境，以保证在多变的环境下不会因为硬件的区别导致项目失败。

（2）软件资源。

在硬件具备的情况下，我们首先要考虑的是软件环境，如测试需要的操作系统、应用软件、管理软件等。像我们平时使用的 Windows、Linux 等操作系统，SQL Server、Oracle 等数据库软件，QuickTest Professional、LoadRunner 等自动化测试工具，其实在当前的网络共享时代，这些都可以找到，不过需要注意版权问题。

（3）技术支持。

有时测试人员在遇到一些问题的时候，并不能靠自身的能力去解决，这需要兄弟部门给予支持，不管是技术还是其他方面的，确保在一个团队合作的环境下，更高效地完

成测试工作，像华为公司在做测试项目时会指派对应的环境保障人员或开发人员作为技术支持。

### 4.1.2　测试工作流程实例

测试部门的工作流程严格意义上来说是按照软件的生命周期作为流转依据，主要有：测试准备阶段、测试开展阶段、测试输出阶段这几个环节，如图 4-4 所示。

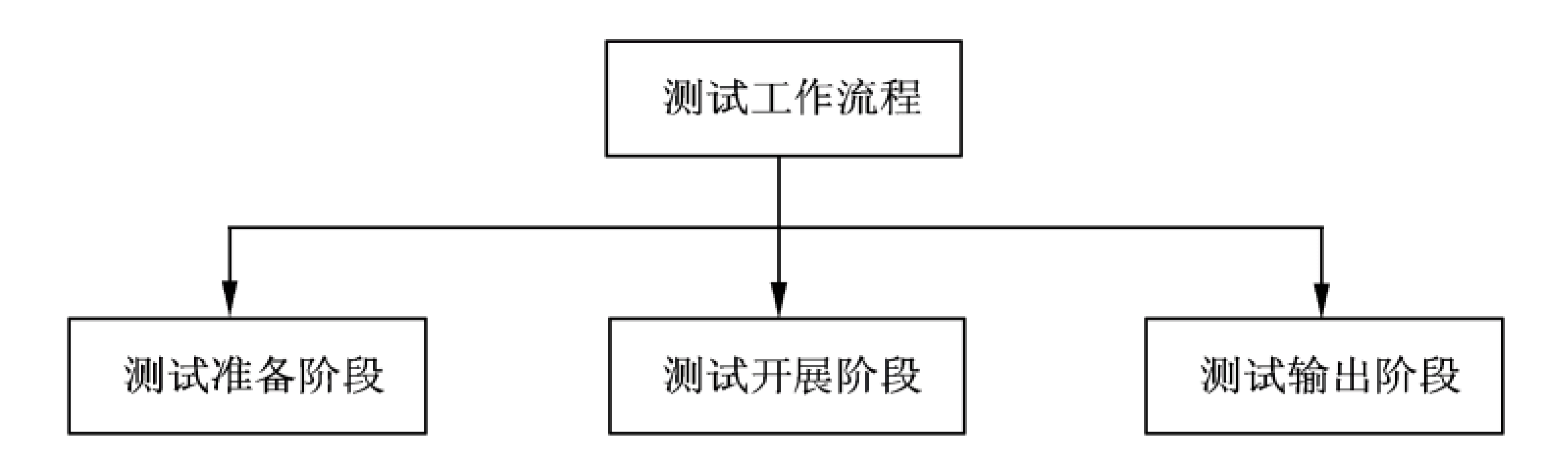



图 4-4　测试工作流程图

**1．测试准备阶段**

在一个项目开展的前期，需要进行需求调研等一系列的准备工作，这时测试部门需要做的事是参与进前期的需求调研（但大多数不会），然后根据需求调研阶段生成的需求说明书指导下一步工作。在这个阶段，主要包括下面几个步骤。

（1）测试计划制订。

在项目立项后，项目经理会根据实际的情况，告知测试部主管，需要相应的测试小组参与进项目中来。那么测试主管会根据部门人员的构成以及技术构成进行协调，指派两名测试组长。由测试组长负责该项目的测试工作。测试组长将会联系项目经理，获取项目的需求规格说明书，然后制订相应的测试计划，安排如何开展本项目的测试工作。具体流程如图 4-5 所示。

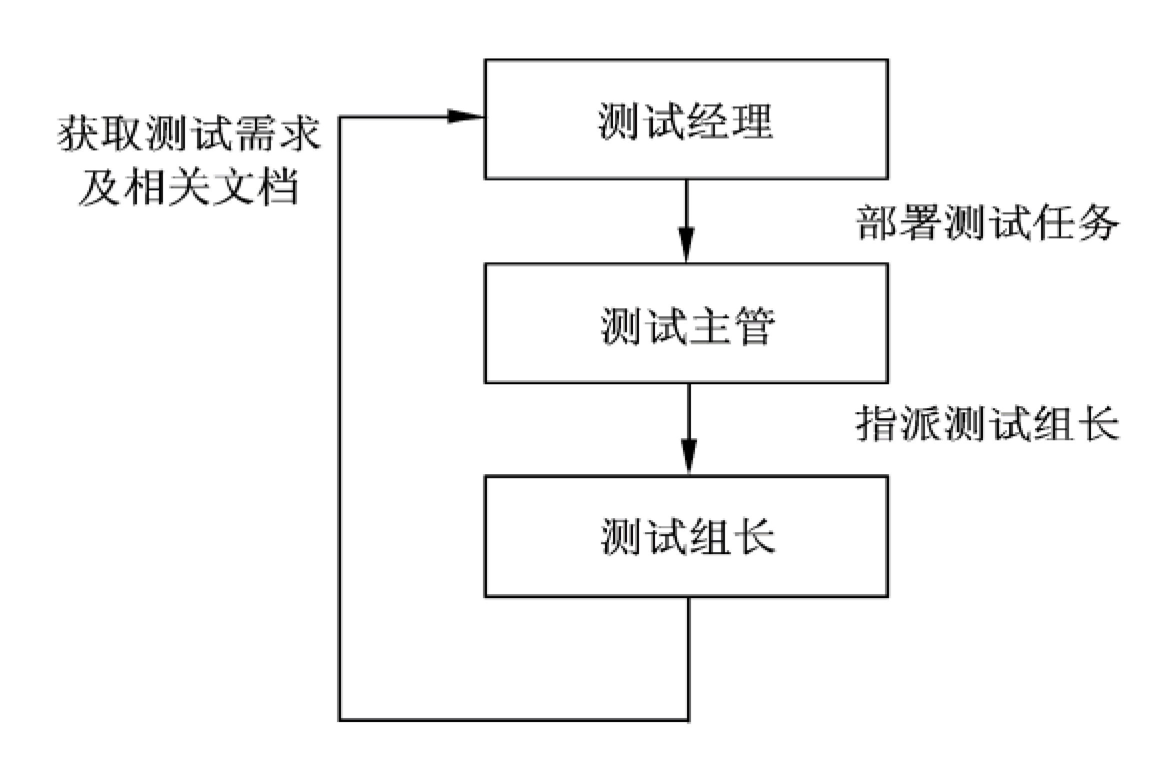



图 4-5　测试工作介入流程图

案例：项目经理张三告知测试主管李四，下周将有一个项目 A 需测试部门进行测试，已经提交了测试申请，请求测试经理安排人员测试。测试主管李四考察项目 A 状况，安排测试组长王五，由王五负责项目 A 测试工作，王五接到工作任务后，会从项目经理处获取项目需求以及相关的系统文档，然后制订该项目的测试计划。此时，测试部门将正式进入到该项目组。

（2）测试小组建立。

在测试组长制订了项目测试计划后，测试组长会根据项目周期长短、项目规模大小组建合适的测试小组，在组建测试小组过程中，多数情况下，小组成员多由测试主管指定。成立测试小组后，组长会召开测试项目的通气会，让组员清晰本项目的相关情况。图 4-6 说明了测试小组的建立流程。

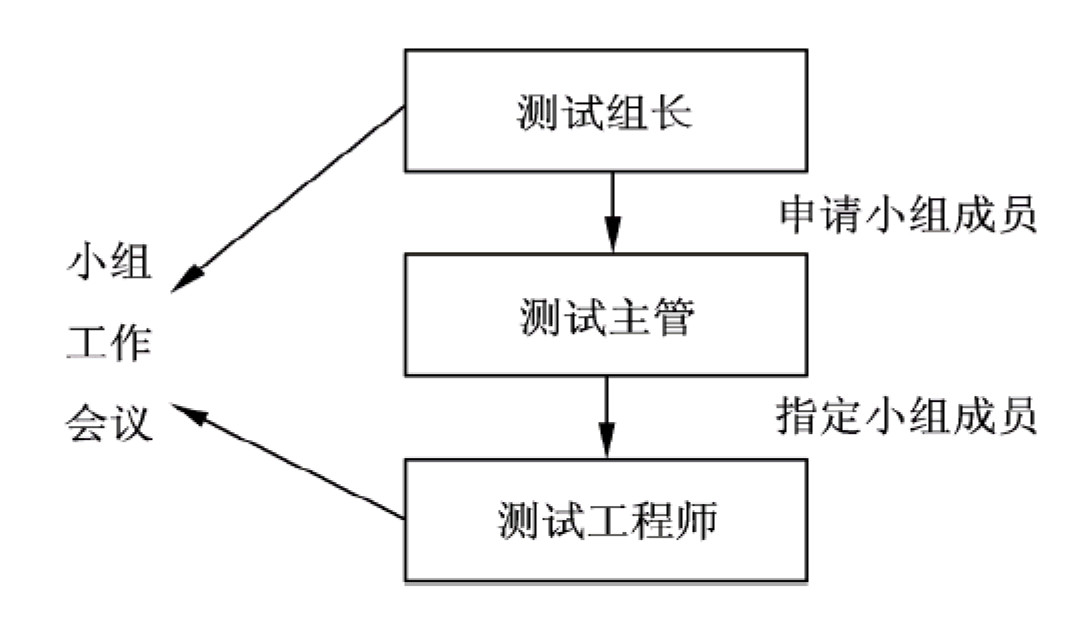



图 4-6 测试小组建立图

案例：项目 A 测试组长王五，根据项目 A 的实际情况，向测试主管李四申请测试组员。测试主管李四，会根据目前部门的工作任务分配情况，指定合适的测试工程师参与进该测试小组。小组建立后，测试组长王五将召开小组会议，介绍当前项目的进展情况以及当前的工作任务。

（3）需求测试启动。

测试小组成立后，测试组长将会安排小组成员阅读需求文档及其他项目文档，开展需求测试工作。可以按照需求的功能结构划分测试任务，亦可整体阅读测试。此阶段，测试工程师需提交需求测试结果报告，并对测试结果报告进行评审，如果合格，则开展测试需求提取工作；如不合格，则由测试组长将需求测试结果反馈至需求文档及其他项目文档的相关提供部门进行校正，校正完成后再次测试，直至合格为止。图 4-7 显示了需求测试的流程图。

案例：测试组长王五安排小组成员阅读项目 A 的需求规格说明书及其他项目文档，并开始需求测试，最后对测试结果进行评审，如果通过，则开始测试需求的提取，如果不通过，则由需求调研部门核实需求，修改后再次测试，直至通过为止。

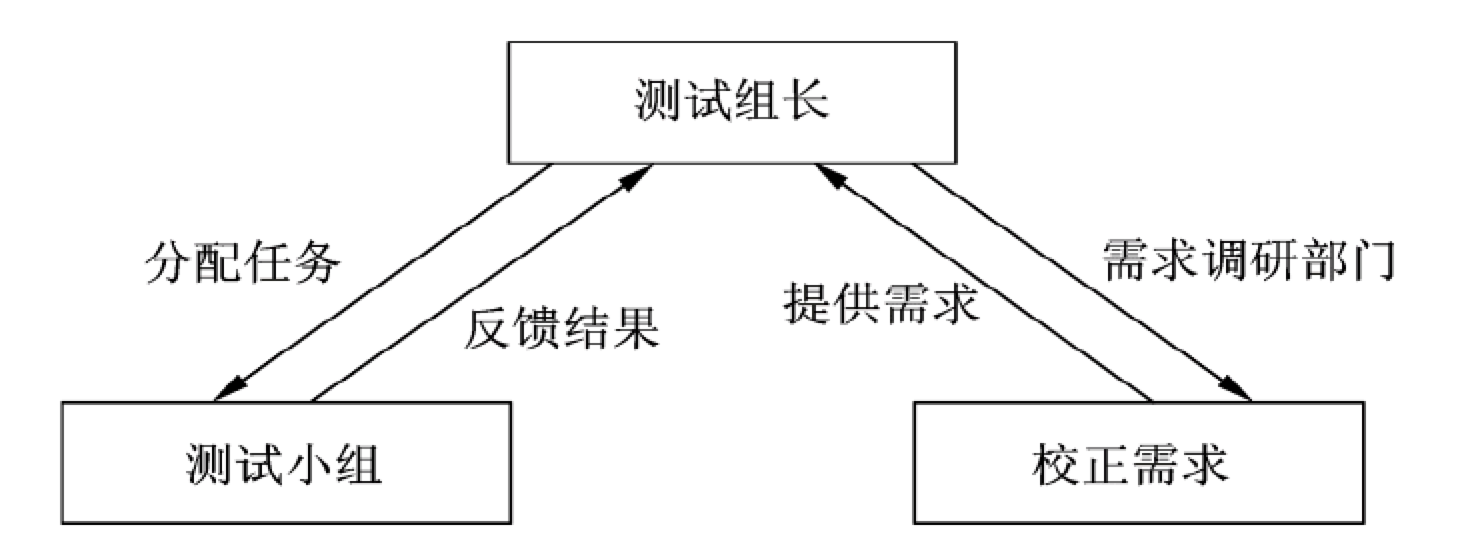



图 4-7　需求测试流程图

（4）测试需求提取。

在需求评审通过后，测试组长将根据测试工程师的技术能力及工作经验，恰当地分配系统功能模块给他们，然后结合公司所使用的测试管理工具，如 TestDirector，进行测试需求的提取，所谓测试需求，通俗点讲，就是我们需要测试的任务点。此阶段，按照正规的工作流程，仍需要进行评审活动，以检查需求提取过程中，是否存在多余、遗漏等错误。评审合格后，开始测试用例编写流程。当然，在这个阶段也可能需要编写测试方案。图 4-8 展示了测试组长部署测试需求提取任务的一般流程。

> 案例：测试组长王五根据小组成员的工作能力，分配相应的系统测试模块，然后利用 TestDirector 进行测试需求的提取和管理。在经过多次评审通过后，将开始测试用例的编写。

（5）测试用例编写。

在测试需求提取工作完成后，我们就开始测试用例的编写。测试用例的编写是个重点和难点。测试用例是开展软件测试的指导性工件。在这个阶段，同样会有多次的测试用例评审会议，检查每个成员所写的测试用例的正确性及效率。同样，在这个阶段可以使用测试管理工具。图 4-9 是部署测试用例任务流程图。

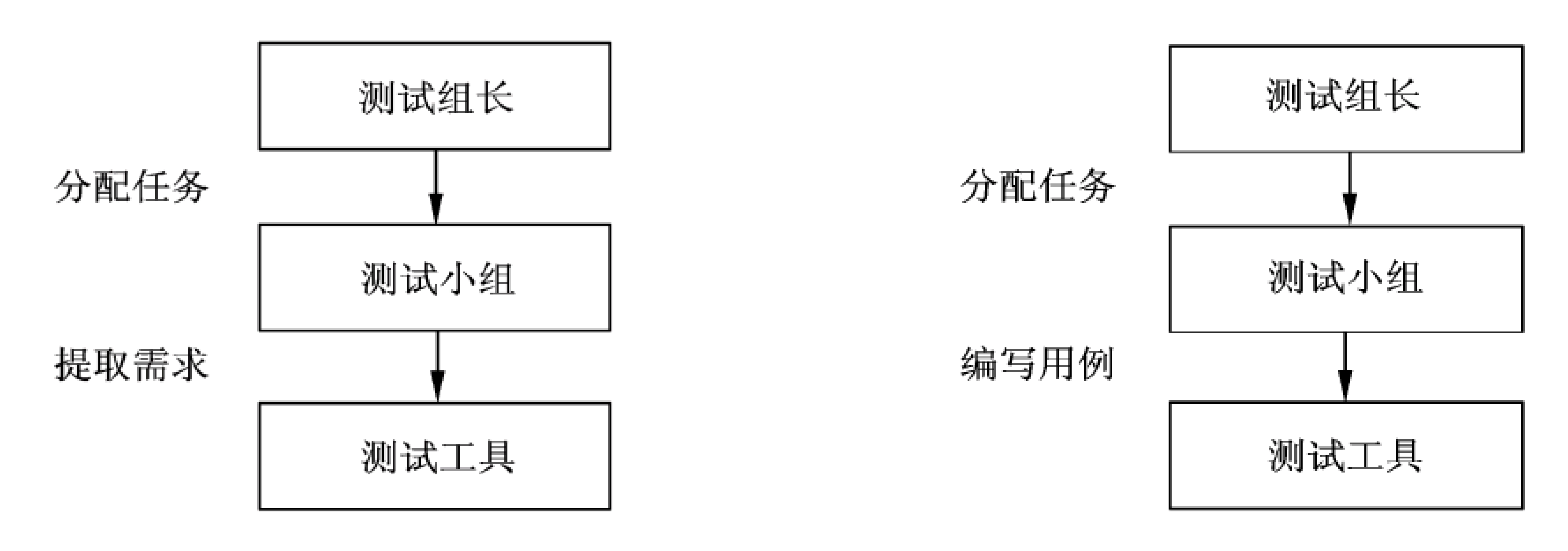



图 4-8　部署测试需求提取任务流程图　　图 4-9　部署测试用例编写任务流程图

> 案例：测试组员根据测试需求设计相应的测试用例，并利用 TestDirector 进行测试用例的管理。经过多次评审后，测试用例应该设计得比较完善了。此时，测试前的准备工作基本上就完成了。

**2．测试开展阶段**

测试准备阶段的任务完成及编码工作完成后，就可以开始正式项目测试工作了。这个过程比较简单，主要是搭建测试环境、文档引入及执行测试 3 个部分。

测试组长将负责搭建测试环境，当然也可以安排小组内其他人员搭建，此时，可根据需求规格说明书中的软件产品运行环境配置要求进行搭建，也可以从开发同事那里获取该软件的环境搭建单。这里需注意的是，测试环境最好与开发环境分开。

接下来，需将本次测试过程中可能使用到的各种文档规划好，并告知小组成员如何使用这些文档，比如每天的工作日报，功能测试报告、性能测试报告等相关文档的模板。最后，就可以执行我们前期设计的测试用例了。根据项目的 Bug 管理流程，经过多次的版本迭代，完成测试工作。

**3．测试输出阶段**

测试工作开展过程中，需要输出很多工件，比如测试计划、测试方案、测试用例、测试工程师工作日报、功能测试报告、性能测试报告等。这些都是软件测试过程中的输出工件。

项目经理会根据最终的软件产品测试报告，来衡量当前软件版本的质量，以决定是否发布。

## 4.2 软件测试项目的过程与步骤

美国 Carnegie Mellon 大学软件工程研究所(Software Engineering Institute)的 Don McAndrews 于 1997 年提出一个软件测试过程(Software Test Process)模型，该测试过程模型可用于系统测试、验收测试或第三方软件测试过程。在此模型的基础上进行适当的扩充形成一个典型软件测试过程模型，该测试过程包括如下 5 个主要活动。

- 测试计划。确定测试基本原则、生成测试概要设计。
- 测试需求分析。
- 测试设计。包括测试用例设计和测试流程设计。
- 测试执行。
- 总结生成报告。

### 4.2.1 测试计划

测试计划活动在软件开发项目的定义、规划、需求分析阶段执行，该项活动确定测试的基本原则并生成测试活动的高级计划。

测试计划在软件项目启动时开始，活动输入是项目进度表和系统/软件功能需求的描述（如软件的需求规格说明）。

测试计划包括以下步骤。

（1）项目经理和测试负责人共同参与测试过程相关的测试需求评审，主要包括：

- 进度表中各阶段的日程。
- 作为测试活动输入的相关合同中的可交付项。
- 项目进度表中针对测试活动而指定的时间。
- 客户指定的测试级别。
- 估计分配给测试活动的小时数。
- 客户在规格说明中指定的质量准则。

（2）测试负责人制定一个针对该项目的测试策略，包括阶段、类型和级别。

（3）测试负责人完成测试计划、用户规格说明、需求验证测试矩阵等测试策略文档，包括由客户规格说明定义的、从单元/集成测试到系统/验收测试的测试级别流程图。

（4）测试负责人标示测试过程中产生的所有产品名称及交付日期。

（5）项目经理和测试负责人标示项目功能需求的来源（如用户需求规格说明、功能规格说明、系统规格说明、合同或其他文档），以便于实现需求追踪。

（6）测试负责人审查用户需求规格说明中的功能需求以确定逻辑测试集。这一工作用于确定可重用策略，如果已存在类似的项目，测试负责人应当审查已有的测试产品以确定这些测试产品能否被重用。

（7）测试负责人书写测试设计规格说明提纲。

（8）测试负责人标示项目中将要进行的所有测试活动，包括测试准备、测试执行和测试后的活动并形成文档。

（9）测试负责人在软件测试计划文档中描述测试活动。

（10）测试负责人在项目进度表中标示测试活动、测试活动的起始和结束时间、风险和不可预见的费用。

（11）测试负责人完成软件测试计划进度表，列出风险和不可预见费用并在测试活动描述中说明其度量。

（12）基于当前可利用资源，测试项目负责人和项目经理安排测试人员和支持测试的人员并写入测试计划中。

（13）测试负责人在软件测试计划文档中描述测试可交付项。

（14）测试负责人和系统工程师定义测试环境（测试环境包括可用的硬件环境和必要的软件）并写入测试检查表中。

（15）测试负责人、配置管理员和系统工程师定义测试控制规程（作为项目配置管理的组成部分）并写入测试计划。

（16）测试负责人根据测试计划模板完成软件测试计划。测试计划包括风险和不可

预见费用、暂停规则、恢复要求、缩略语列表等。

（17）测试计划完成的标志是生成经过评审的软件测试计划文档，软件测试计划应获得客户的认可。测试规划完成后测试进入需求分析阶段。

### 4.2.2 测试需求分析

在确定需求追踪矩阵、完成软件测试计划文档后即进入测试需求分析阶段。测试需求分析活动包括如下步骤。

（1）测试负责人和测试工程师审查需求追踪矩阵中每个需求并为其确定测试方法。

（2）测试负责人和测试工程师审查所有可测的需求并分配到测试设计规格说明中进行详细描述。

（3）任何未在软件测试计划中标示的测试设计规格说明内容应当添加到需求追踪矩阵中。

（4）测试工程师生成关于测试需求指定到测试方法的报告供评审。

（5）测试工程师生成关于可测试需求指定到测试设计规格说明的报告。

（6）随着需求中问题的出现，测试负责人或测试工程师需书面描述问题并与项目经理讨论这些问题。如有必要，会生成基于缺陷的问题报告。

测试需求分析阶段的产品是被批准的需求测试矩阵。

### 4.2.3 测试设计

完成需求测试矩阵和测试计划后，即可进入测试设计阶段，该阶段活动步骤如下。

（1）测试工程师审查客户规格说明、需求可测试矩阵和开发文档确保测试设计规格说明的大纲是恰当的。如果考虑可追踪性将是比较困难的，则选择最具有综合性的文档用于软件的开发和维护（可能是软件需求规格说明或其他文档）。测试用例和规程设计应遵循此大纲，以保证需求的可追踪性。

（2）测试工程师根据测试计划生成测试设计规格说明。

（3）测试工程师审查从需求测试矩阵分配到测试和设计规格说明的每一测试要求，并给出测试用例的逻辑集大纲。

（4）测试工程师根据测试计划生成测试用例规格说明。

（5）测试工程师根据测试用例分配和可追踪的信息更新需求测试矩阵。

（6）测试工程师审查从更新的需求测试矩阵分配到测试和设计规划说明的每一测试要求，并给出测试用例的逻辑集大纲。

（7）测试工程师准备所有测试需求说明和测试用例规格说明，并更新需求测试矩阵用于发布和评审。

本活动完成的标志是生成批准的测试设计规格说明。

### 4.2.4　测试执行

在完成测试流程后进入测试执行阶段，该阶段活动步骤如下。

（1）在测试之前，测试负责人和测试工程师为即将进行的测试，准备执行检查表。

（2）测试工程师确保所有的用例都经过评审和更新。

（3）测试工程师、系统工程师、开发工程师协同工作，为测试事件建立基线和实验设置。所有人都必须知道哪些内容属于基线的范围。

（4）测试工程师和客户或质量管理员一起执行测试。

（5）根据测试事件生成软件问题报告。

（6）测试工程师按照测试计划的定义准备软件测试报告。

### 4.2.5　总结生成报告

在完成测试执行活动后进入总结生成报告阶段。

该活动主要任务是测试负责人根据测试计划、测试流程和软件问题报告，分析测试执行结果，总结生成软件测试报告。活动的结束标志是生成软件测试报告。

# 第 5 章　软件缺陷与缺陷报告

**学习目标：**

1．了解什么是软件缺陷

2．熟悉软件缺陷报告的书写

3．了解缺陷管理跟踪系统

## 5.1　什么是软件缺陷

软件测试是为了发现错误而执行程序的过程，而更多专家认为软件测试的范畴应当更为广泛，除了要考虑测试结果的正确性以外，还应关心程序的效率、可适用性、维护性、可扩充性、安全性、可靠性、系统性能、系统容量、可伸缩性、服务可管理性、兼容性等因素。随着人们对软件测试更广泛、更深刻的认识，可以说对软件质量的判断绝不只限于程序本身，而是整个软件研制过程。

不管怎么定义软件测试，基本的结论是一致的，即软件测试是为了发现软件产品所存在的任何意义上的软件缺陷（Bug），从而纠正（Fix）这些软件缺陷，使软件系统更好地满足用户的需求。那么，什么是软件缺陷呢？

### 5.1.1　缺陷的定义

通过第 2 章实例中所出现的软件问题，在软件工程或软件测试中都被称为软件缺陷或软件故障。在不引起误解的情况下，不管软件存在问题的规模和危害的大小，由于都会产生软件使用上的各种障碍，所以将这些问题统称为软件缺陷。

软件缺陷，即计算机系统或者程序中存在的任何一种破坏正常运行能力的问题、错误或者隐藏的功能缺陷、瑕疵。缺陷会导致软件产品在某种程度上不能满足用户的需要。在 IEEE 1983 of IEEE Standard 729 中对软件缺陷下了一个标准的定义，如下：

从产品内部看，软件缺陷是软件产品开发或维护过程中所存在的错误、毛病等各种问题；从外部看，软件缺陷是系统所需要实现的某种功能的失效或违背。因此软件缺陷就是软件产品中所存在的问题，最终表现为用户所需要的功能没有完全实现，没有满足用户的需求。

### 5.1.2　缺陷的种类

软件缺陷表现的形式有多种，不仅仅体现在功能的失效方面，还体现在其他方面。

软件缺陷的主要类型通常有以下几种。

（1）软件未达到产品说明书中已经标明的功能。

（2）软件出现了产品说明书中指明不会出现的错误。

（3）软件未达到产品说明书中虽未指出但应当达到的目标。

（4）软件功能超出了产品说明书中指出的范围。

（5）软件测试人员认为软件难以理解、不易使用，或者最终用户认为该软件使用效果不良。

为了对以上 5 条描述进行理解，这里以日常我们所使用的计算器内的嵌入式软件来说明上述每条定义的规则。

计算器说明书一般声称该计算器将准确无误地进行加、减、乘、除运算。如果测试人员或用户选定了两个数值后，随意按下了“+”号键，结果没有任何反应或得到一个错误的结果，根据第一条规则，这是一个软件缺陷；如果得到错误答案，根据第一条规则，同样是软件缺陷。

假如计算器产品说明书指明计算器不会出现崩溃、死锁或者停止反应，而在用户随意按、敲键盘后，计算器停止接受输入或没有了反应，根据第二条规则，这也是一个软件缺陷。

若在测试过程中发现，因为电池没电而导致了计算不正确，但产品说明书未能指出在此情况下应如何进行处理，根据第三条规则，这也应算作软件缺陷。

若在进行测试时，发现除了规定的加、减、乘、除功能之外，还能够进行求平方根的运算，而这一功能并没有在说明书的功能中规定，根据第四条规则，这也是软件缺陷。

第五条的规则说明了无论测试人员或者是最终用户，若发现计算器某些地方不好用，比如，按键太小、显示屏在亮光下无法看清等，也都应算作软件缺陷。

软件缺陷一旦被发现，就要设法找出引起这个缺陷的原因，分析对产品质量的影响，然后确定软件缺陷的严重性和处理这个缺陷的优先级。各种软件缺陷所造成的后果是不同的，有的仅仅是不方便，有的则可能是灾难性的。一般来说，问题越严重的，其优先级越高，越要得到及时的纠正。软件公司对缺陷严重性级别的定义不尽相同，但一般可概括为以下几种。

（1）致命的：致命的错误，造成系统或应用程序崩溃、死机、系统悬挂，或造成数据丢失、主要功能完全丧失等。

（2）严重的：严重错误，指功能或特性没有实现、主要功能丧失、会导致严重的问题或致命的错误声明。

（3）一般的：不太严重的错误，这样的软件缺陷虽然不影响系统的基本使用，但没有很好地实现功能，没有达到预期效果。如次要功能丧失、提示信息不太准确、用户界面差、操作时间长等。

（4）微小的：一些小问题，对功能几乎没有影响，产品及属性仍可使用，如有个别

错误字、文字排列不整齐等。

除了这 4 种以外，有时需要“建议”级别来处理测试人员所提出的建议或质疑，对建议程序做适当的修改，来改善程序运行状态，或对设计不合理、不明白的地方提出质疑。

### 5.1.3 缺陷的产生

软件缺陷的产生是不可避免的，那么造成软件缺陷的原因是什么呢?通过大量的测试理论研究及测试实践经验的积累，软件缺陷产生的主要原因可以被归纳为以下几种类型。

（1）需求解释有错误。

（2）用户需求定义错误。

（3）需求记录错误。

（4）设计说明有误。

（5）编码说明有误。

（6）程序代码有误。

（7）其他，如数据输入有误，问题修改不正确。

由此可见，造成软件缺陷的原因是多方面的。经过软件测试专家们的研究发现，大多数的软件缺陷并非来自编码过程中的错误，从小项目到大项目都基本上证明了这一点。因为软件缺陷很可能是在系统详细设计阶段、概要设计阶段，甚至是在需求分析阶段就存在着问题，即使是针对源程序进行的测试所发现的故障的根源也可能存在于软件开发前期的各个阶段。大量的事实表明，导致软件缺陷的最大原因是软件产品说明书，也是软件缺陷出现最多的地方。

在多数情况下，软件产品说明书写得不明确、不清楚、描述不全面，或者在软件开发过程中对需求、产品功能经常更改，或者开发小组的人员之间没有很好地进行交流与沟通，没有很好地组织开发与测试流程。因此，制作软件产品开发计划是非常重要的，如果计划没有做好，软件缺陷就会出现。

软件缺陷产生的第二大来源是设计方案，这是实施软件计划的关键环节。

编程排在第三位。许多人认为软件测试主要是找程序代码中的错误，这是一个认识的误区。经统计，因编写程序代码引入的软件缺陷大约仅占缺陷总数的 7%。

### 5.1.4 软件缺陷的分布

根据上面讨论，我们知道软件缺陷是由很多原因造成的，如果把它们按需求规格说明书、系统设计结果、编程的代码等归类起来，比较后发现，结果需求规格说明书是软件缺陷出现最多的地方，如图 5-1 所示。

软件产品需求规格说明书为什么是软件缺陷存在最多的地方，主要原因有以下

几种。

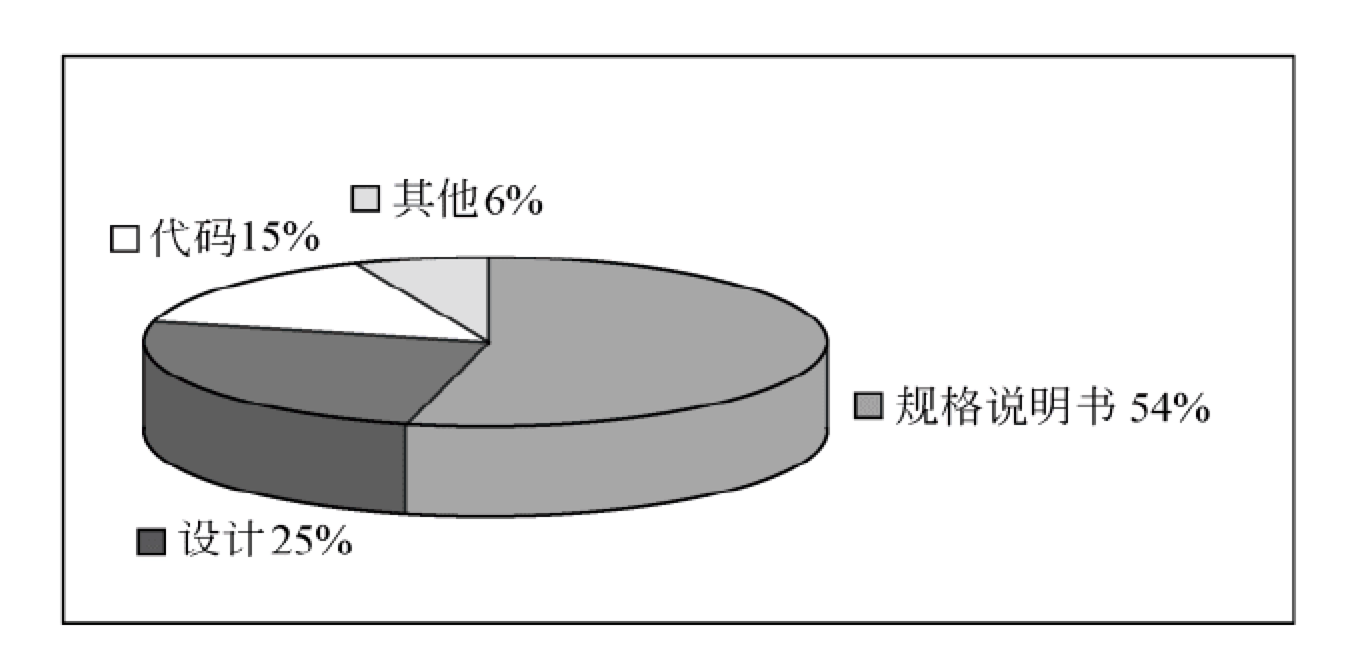



图 5-1　软件缺陷构成示意图

（1）用户一般是非计算机专业人员，软件开发人员和用户的沟通存在较大困难，对要开发的产品功能理解不一致。

（2）由于软件产品还没有设计、开发，完全靠想象去描述系统的实现结果，所以有些特性还不够清晰。

（3）需求变化的不一致性。用户的需求总是在不断变化的，这些变化如果没有在产品规格说明书中得到正确的描述，容易引起前后文、上下文的矛盾。

（4）对规格说明书不够重视，在规格说明书的设计和写作上投入的人力、时间不足。

（5）没有在整个开发队伍中进行充分沟通，有时只有设计师或项目经理得到比较多的信息。

排在产品规格说明书之后的是设计，编程排在第三位。在许多人的印象中，软件测试主要是找程序代码中的错误，这是一个认识的误区。

### 5.1.5　修复软件缺陷的代价

在一开始讨论软件测试时，我们曾说测试人员要从需求分析时就介入进去，问题发现得越早越好。缺陷被发现之后，要尽快修复这些被发现的缺陷。为什么要这样做呢？原因很简单，错误并不只是在编程阶段产生，需求和设计阶段同样会产生错误。也许一开始，只是一个很小范围内的潜在错误，但随着产品开发工作的进行，小错误会扩散成大错误，为了修改后期的错误所做的工作量要大得多。缺陷发现或解决得越迟，成本就越高。

Boehm 在 *Software Engineering Economics*（1981 年）一书中写道：平均而言，如果在需求阶段修正一个错误的代价是 1，那么，在设计阶段就是它的 3～6 倍，在编程阶段是它的 10 倍，在内部测试阶段是它的 20～40 倍，在外部测试阶段是它的 30～70 倍，而到了产品发布出去时，这个数字就是 40～1000 倍。修正错误的代价不是随着时间线性增长，而几乎是呈指数增长的。图 5-2 就是说明这样一个道理。

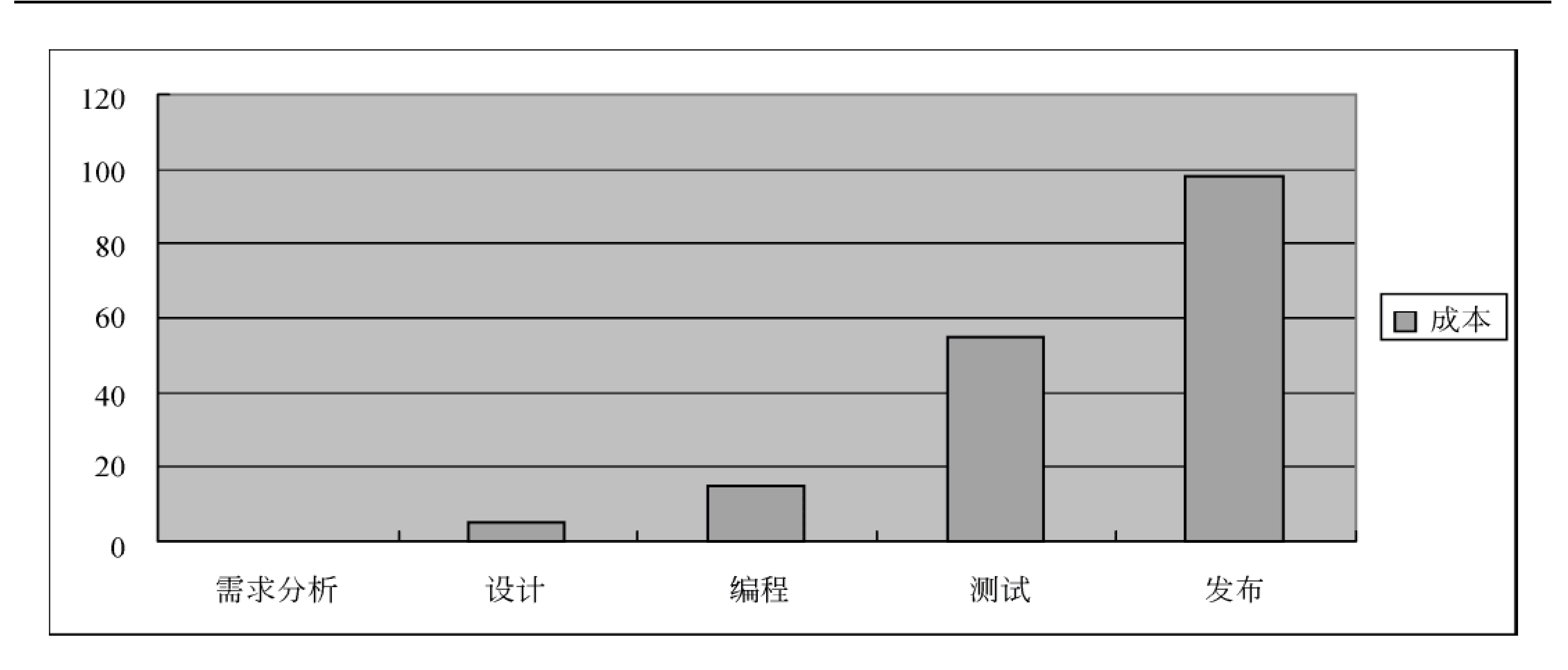



图 5-2 软件缺陷随着时间的推移带来的成本越来越大

## 5.2 怎样报告软件缺陷

软件测试人员使用软件缺陷跟踪系统的一项主要工作是编写软件缺陷报告。提供准确、完整、简洁、一致的缺陷报告是体现软件测试的专业性、高质量的评价指标之一。如果缺陷报告包含过少或过多信息，组织混乱，则很难确认该缺陷，由此导致缺陷被退回，延误修正，或者由于没有清楚地说明缺陷的影响而使这些缺陷随软件版本一起发布出去。

为了提高缺陷报告的质量，需要明确缺陷报告的读者对象，遵守书写缺陷报告的通用规则，合理组织缺陷报告的格式结构，掌握常用的缺陷报告技术。

### 5.2.1 谁会阅读缺陷报告

在书写软件缺陷报告之前，需要明白谁会阅读缺陷报告，了解读者最希望从缺陷报告中获得什么信息。通常，缺陷报告的直接读者是软件开发人员和质量管理人员，除此之外，来自市场和技术支持等部门的人也可能需要查看缺陷情况。每个阅读缺陷报告的人都需要理解缺陷针对的产品和使用的技术。另外，他们不是软件测试人员，可能对于具体软件测试的细节了解不多。

概括起来，缺陷报告的读者最希望获得的信息包括以下几点。

（1）易于搜索软件测试报告的缺陷。

（2）对报告的软件缺陷进行了必要的隔离，报告的缺陷信息更具体、准确。

（3）软件开发人员希望获得缺陷的本质特征和复现步骤。

（4）市场和技术支持等部门希望获得缺陷类型分布以及对市场和用户的影响程度。

软件测试人员的任务之一就是需要针对读者的上述要求，书写良好的软件缺陷报告。

### 5.2.2　写好缺陷报告的重要性

有些测试人员认为录入的 Bug 描述不清晰不要紧，如果导致开发人员误解的话，开发人员应该主动询问测试人员。这话有一定的道理，这确实存在沟通上的问题。但是测试人员如果尽量清晰地描述缺陷，尽量让开发人员一看就明白是什么问题，甚至明白是什么原因引起的错误，这样可节省更多沟通上的时间。

需要引起测试人员注意的是，Bug 的质量除了缺陷本身外，描述这个 Bug 的形式载体也是其中一种衡量标准。如果把测试人员发现的一个目前为止尚未出现的高严重级别的 Bug 称之为一个好 Bug，那么录入的 Bug 描述不清晰，令人费解，难以按照描述的步骤重现的话，则会大大地有损这个好 Bug 的“作用”。

### 5.2.3　书写缺陷报告的基本规则

书写清晰、完整的缺陷报告是保证正确处理缺陷的最佳手段。它也减少了工程师以及其他质量保证人员的后续工作。为了书写更优良的缺陷报告，需要遵守“5C”准则。

（1）**Correct（准确）**：每个组成部分的描述准确，不会引起误解。

（2）**Clear（清晰）**：每个组成部分的描述清晰，易于理解。

（3）**Concise（简洁）**：只包含必不可少的信息，不包括任何多余的内容。

（4）**Complete（完整）**：包含复现该缺陷的完整步骤和其他本质信息。

（5）**Consistent（一致）**：按照一致的格式书写全部缺陷报告。

### 5.2.4　组织结构

尽管不同的软件测试项目对于缺陷报告的具体组成部分不尽相同，但是基本组织结构都是大同小异的。一个完整的软件缺陷报告通常由下列几部分组成。

（1）缺陷的标题。

（2）缺陷的基本信息，包括以下几方面。

① 测试的软件和硬件环境。

② 测试的软件版本。

③ 缺陷的类型。

④ 缺陷的严重程度。

⑤ 缺陷的处理优先级。

（3）复现缺陷的操作步骤。

（4）缺陷的实际结果描述。

（5）期望的正确结果描述。

（6）注释文字和截取的缺陷图像。

对于具体测试项目而言，缺陷的基本信息通常是比较固定的，也是很容易描述的。实际书写软件缺陷报告容易出现问题的地方就是标题、操作步骤、实际结果、期望结果和注释部分。下面针对这些“事故多发地带”，具体论述如何提供完整的信息。由于英文是软件开发的主要语言，以下的软件缺陷报告的信息都使用英文书写。

## 5.2.5 写作技术

**1. 标题（Title）**

标题应该保持简短、准确，提供缺陷的本质信息，并且便于读者搜索查寻。良好的缺陷标题应该按照下列方式书写。

（1）尽量按缺陷发生的原因与结果的方式书写（“执行完 A 后，发生 B”或者“发生 B，当 A 执行完后”）。

（2）避免使用模糊不清的词语，如“功能中断，功能不正确，行为不起作用”等。应该使用具体文字说明功能如何中断，如何不正确，或如何不起作用。

（3）为了方便搜索和查询，请使用关键字。

（4）为了便于他人理解，避免使用术语、俚语或过分具体的测试细节。

请查看表 5-1，该表列出了有问题的标题，给出了如何改进的示例。

**表 5-1 缺陷标题的描述**

| 原 始 描 述 | 错 误 原 因 | 改进的标题 |
|---|---|---|
| “Hyphenation does not work” | 描述太笼统。什么时候不起作用? | “Text breaks at line’s end, but no hyphen appears” |
| “Incorrect behavior with paragraph alignment” | 描述太笼统。不正确的行为是什么? | “Justified alignment leaves gaps in text composition when tracking is also applied” |
| “Assert：Cmd Assert Here insert Something Bad Happens” | 没有包含原因与结果信息。断言（Assert）太长。 | “Assert，Something Bad when attempting to update linked bitmap stored on server” |
| “After each launch then clicking edit and then copy/paste，there is too much delay” | 没有指明原因与结果，包含了过分详细的细节信息 | “Performance slows noticeably after first launch and copy/paste” |
| “Quotes appear as symbols when they are imported” | 信息没有充分隔离。所有的引号都如此吗?什么类型的符号等等? | “Imported smart quotes from Word Appear as unrecognized characters” |

使用“after”，“when”或“during”等连结词有助于描述缺陷的原因和结果，例如：

- Application crashes after inputting any letters in numeric field.
- Internal error occurs when closing application.
- Application suspended during email transmission.

**2．复现步骤（Reproducible Steps）**

复现步骤包含如何使别人能够很容易地复现该缺陷的完整步骤。为了达到这个要求，复现步骤的信息必须是完整的、准确的、简明的、可复现的。

但是实际软件测试过程中，总是存在一些不良的缺陷报告，主要的问题在于以下 3 个方面。

（1）复现步骤包含了过多的多余步骤，而且句子结构混乱，可读性很差，难于理解。

（2）复现步骤包含的信息过少，丢失了操作的必要步骤。由于提供的步骤不完整，开发人员经常需要猜测，努力尝试复现的步骤，浪费了大量时间。由于缺少关键步骤，这些缺陷通常被工程师以“不能复现”为由再次发送给测试人员。

（3）测试人员没有对软件缺陷发生的条件和影响区域进行隔离，软件开发人员无法判断该缺陷影响的软件部分，不能进行彻底修正。

为了避免出现这些问题，良好的复现步骤应该包含本质的信息，并按照下列方式书写。

（1）提供测试的预备步骤。

- 环境变量。例如，如果默认项或预设、调试版本或发布版本等存在问题，请指明使用的操作系统和应用程序的环境变量。
- 设置变量：指明哪种打印机、字体或驱动程序是复现 bug 所必需的信息。
- 复现路径。如果有多种方法触发该缺陷，请在步骤中包含这些方法。同样地，如果某些路径不触发该缺陷，也要包含这些路径。

（2）简单地一步一步地引导复现该缺陷。

（3）每一个步骤尽量只记录一项操作。

（4）每一个步骤前使用数字对步骤编号。

（5）尽量使用短语和短句，避免复杂句型、句式。

（6）复现的操作步骤要完整、准确、简短，以保证：

- 不要缺漏任何操作步骤。
- 每个步骤都是准确无误的。
- 没有任何多余的步骤。

（7）将常见步骤合并为较少步骤。

（8）只记录各个操作步骤是什么，不要包括每个操作步骤执行后的结果。

**3．实际结果**

实际结果是执行复现步骤后软件的现象和产生的行为。

实际结果的描述很像缺陷的标题，是标题信息的再次强调，要列出具体的表现行为，而不是简单地指出“不正确”或“不起作用”。

如果一个动作产生彼此不同的多个缺陷结果，为了易于阅读，这些结果应该使用数字列表分隔开来。有时，一个动作将产生一个结果，而这个结果又产生另一个结果。这种情况可能难以清晰、简洁地总结出来。

对于这些较难处理的情况，有多种使之易于阅读的解决方法，列举如下：

（1）尽可能将缺陷分解成多个缺陷报告，并使用交叉引用说明彼此之间的联系。这些动作的结果通常比较相似但缺陷不同。首先进行更多的隔离测试，缩小产生缺陷的范围，查看是否产生多个缺陷。

（2）在实际结果部分，仅列出缺陷的一到两个表现特征。使用注释（Notes）部分列出缺陷的其他问题。

在缺陷的第一个表现特征后，将随后的步骤和缺陷表现特征移到注释部分。重要的信息几乎总是包含在第一个断言或错误里，其他错误都是第一个错误的变种。

**4．期望结果（Expected Result）**

期望结果的描述应该与实际结果的描述方式相同。通常需要列出期望的结果应该是什么，并且给出期望结果的原因，可能是引用的规格说明书、前一版本的表现行为、客户的一般需求、排除杂乱信息的需要等等。

为了更清楚地理解良好的期望结果应该包含什么信息，请看下面的例子（见图 5-3）。

**Expected result:**

The text that appears should be fully highlighted so that if the user wishes to make changes, they don’t have to manually highlight and then type（as in Mac OS 10. x and Windows behavior）.

图 5-3 期望结果

这个期望结果的描述内容准确、丰富、有理有据。

- 应该产生的正确现象：The text that appears should be fully highlighted。
- 为什么应该产生：...so that if the user wishes to make changes，they don’t have to manually highlight and then type。
- 给出了具体的参考对象：As in OS 10． x and Windows behavior。

**5．注释（Notes）**

注释应该包括复现步骤中可能引起混乱的补充信息，是对操作步骤的进一步描述，这些补充信息是复现缺陷或隔离缺陷的更详细的内容。

注释部分可以包含以下方面的内容。

（1）截取缺陷特征图像文件（Screenshots）。

（2）测试过程需要使用的测试文件。
（3）测试附加的打印机驱动程序。
（4）再次描述重点，避免开发人员将缺陷退回给测试人员补充更多信息。

- 再次指明该缺陷是否在前一版本已经存在。
- 多个平台之间是否具有不同表现。

（5）注释包含缺陷的隔离信息。

### 5.2.6 缺陷报告的写作要点

**1. 自我检查和提问**

（1）缺陷报告已经向读者提供了包含完整、准确、必要的信息了吗?
（2）一个缺陷报告中是否只报告了一种缺陷?
（3）读者是否能容易地搜索该缺陷?
（4）步骤是否可以完全复现而且表达清楚?
（5）是否包含了复现该缺陷需要的环境变量或测试所用的数据文件?
（6）缺陷的标题是按照原因与结果的方式书写的吗?
（7）实际结果和期望结果是否描述不够清楚而容易引起歧义?

**2. 避免常见的错误**

（1）使用“I”（我）、“You”（你）等人称代词。可以直接使用动词或必要时使用“User”（用户）代替。

（2）使用情绪化的语言和强调符号，例如黑体、全部字母大写、斜体、感叹号、问号等。只要客观地反映出缺陷的现象和完整信息即可，不要对软件的质量优劣做任何主观性强烈的批评、嘲讽。

（3）使用诸如“Seems”（似乎）、“Appears to be”（看上去可能）等含义模糊的词汇，而需要报告确定的缺陷结果。

（4）使用自认为比较幽默的语句，因为不同读者的文化和观念不同，很多幽默语句在别人看来，往往难以准确理解，甚至可能引起误解。只需客观地描述缺陷的信息即可。

（5）将不确定的测试问题（Issues）放在缺陷管理数据库中。如果对测试软件的某个现象不确定是否是软件缺陷，可以通过电子邮件或口头交流，确认是缺陷后再报告到数据库中。避免查询和统计结果的不准确性。

### 5.2.7 缺陷报告应该注意的问题

Bug 报告是测试人员辛勤劳动的结晶，是测试人员价值的体现，同时也是与开发人员交流的基础。Bug 报告是否正确、清晰与完整将直接影响开发人员修改 Bug 的效率和质量，因此，在报告 Bug 时，需要注意以下几个问题。

**1. 尽量避免出现错误**

测试人员会经常找出开发人员关于界面上的错别字、用词不当、描述不清楚等错误，但是，测试人员在录入 Bug 的时候却很容易犯同样的错误，这种情况在测试人员中应该尽量避免发生，正所谓“己所不欲，勿施于人”。

**2. 不要把几个 Bug 录入到同一个 ID**

即使一些 Bug 的表面现象类似，或者是一些 Bug 同时出现在同一个区域，也应该一个缺陷对应录入一个 Bug。因为这样才能清晰地跟踪所有 Bug 的状态，并且有利于缺陷的统计和质量的衡量。

**3. 添加必要的截图和文件**

一些涉及用户界面（User Interface）的软件缺陷可能很难用文字清楚地描述，所谓“一图胜千言”，把错误的界面截取下来，添加到 Bug 报告中，可以让开发人员清楚地看到 bug 出现时的情形。最好能在截图中用画笔圈出需要注意的地方，这样能直观地表示缺陷发生在产品界面什么位置、有什么问题等。

附件中添加截图是现在 Bug 报告中最常见的形式，下面我们为大家详细介绍一下添加截图的一些规则：

（1）采用图片的格式。

测试人员一般采用 JPG、GIF 的图片格式，因为这类文件占用的空间小，打开的速度快。

（2）什么情况下需要附上图片。

通常情况下，出现在用户界面（User Interface），并且影响用户使用或者影响产品的美观的软件缺陷，附上图片比较直观，例如：

① 当产品中有一段文字没有显示完全，为了明确标识这段文字的位置，测试人员必须贴上图片。

② 在测试外国语言版本的时候，当发现产品中有一段文字没有翻译，测试人员需要贴上图片标识没有翻译的文字。

③ 在测试外国语言版本的时候，当发现产品中有一段外国文字显示乱字符，测试人员必须贴上图片标识那些乱字符的外国文字。

④ 产品中的语法错误、标点符号使用不当等软件缺陷，测试人员贴上图片告诉开发人员缺陷在什么地方。

⑤ 在产品中运用错误的公司标志和重要的图片没有显示等软件缺陷，也需要附上图片。

注意：必要的异常信息文件、日志文件、输入数据文件也可作为附件添加到 Bug 报告中，以方便开发人员定位和重现错误。

**4．完成一 Bug 的录入后应进行检查**

就像要求程序员在编写完代码后应自己编译并做初步的测试一样，应该要求测试人员在录入完一个 Bug 后自己阅读一遍，检查语句是否通顺，表达是否清晰。

## 5.3　软件缺陷跟踪管理

软件测试的主要目的在于发现软件存在的错误（Bug），如何处理测试中发现的错误，将直接影响到测试的效果。只有正确、迅速、准确地处理这些错误，才能消除软件错误，保证要发布的软件符合需求设计的目标。在实际的软件测试过程中，每个 Bug 都要经过测试、确认、修复、验证等的管理过程，这是软件测试的重要环节。

**1．错误跟踪管理**

为了正确地跟踪每个软件错误的处理过程，通常将软件测试发现的每个错误作为一条记录输入指定的错误跟踪管理系统。

目前已有的错误跟踪管理软件包括 Compuware 公司的 TrackRecord 软件（商业软件）、Mozilla 公司的 Bugzilla 软件（免费软件），以及国内的微创公司的 BMS 软件，这些软件在功能上各有特点，可以根据实际情况选用。当然，也可以自己开发缺陷跟踪软件，例如基于 Notes 或是 ClearQuese 开发的错误跟踪管理软件。

作为一个错误跟踪管理系统，需要正确记录错误信息和错误处理信息的全部内容。

（1）Bug 记录信息。

主要包括以下几项内容。

- 测试软件名称。
- 测试版本号。
- 测试人名称。
- 测试事件。
- 测试软件和硬件配置环境。
- 发现软件错误的类型。
- 错误的严重等级。
- 详细步骤。
- 必要的附图。
- 测试注释。

（2）Bug 处理信息。

主要包括以下 4 项内容。

- 处理者姓名。
- 处理时间。
- 处理步骤。

- 错误记录的当前状态。

正确的错误数据库权限管理是错误跟踪管理系统的重要考虑要素，一般要保证对于添加的错误不能从数据库中删除。

**2. 软件错误的状态**

软件错误的主要状态包括以下内容。

- 新建（NEW）：测试中新报告的软件 Bug。
- 已分配（ASSIGNED）：被确认并分配给相关开发人员处理。
- 已解决（RESOLVED）：开发人员已完成修正，等待测试人员验证。
- 重新打开（REOPENED）：测试人员返测状态为已解决的缺陷，开发人员已修改完成的缺陷，发现所描述的错误没有改正，需要重新打开。
- 已验证（VERIFIED）：返测通过，确认缺陷已被修正。
- 关闭（CLOSED）：Bug 已被修复。

**3. 缺陷处理方法**

对于缺陷有以下几种处理的方法。

- 已修复（FIXED）：开发人员已将缺陷修复。
- 无效（INVALID）：开发人员认为不是错误，所提交的缺陷不用修复。
- 暂时不改（WONTFIX）：开发人员任意修改这个缺陷可能会影响到其他方面，需要在以后来修改，或者为修复此缺陷需要提供更多的信息。
- 无法重现（WORKSFORME）：开发人员根据测试人员提交的 Bug 步骤，无法再现 Bug。
- 重复（DUPLICATE）：标记缺陷为另一缺陷的重复。

**4. 错误管理流程**

错误管理的流程可以概括为以下几项内容。

- 测试人员提交新的错误入库，错误状态为“NEW”（新建）。
- 高级测试人员验证错误。
  - ✓ 如果确认是错误，分配给相应的开发人员，设置状态为“ASSIGNED”（已分配）。
  - ✓ 如果不是错误，则视为无效，设置为“INVALID”（无效）状态。
  - ✓ 开发人员查询状态为“ASSIGNED”（已分配）的错误，做如下处理。
  - ✓ 如果不是错误，则置状态为“INVALID”（无效）。
  - ✓ 如果是错误，则修复并置状态为“FIXED”（已修复）。
  - ✓ 如果不能解决的错误，要留下文字说明并保持错误为“WONTFIX”（暂不修改）状态。
  - ✓ 对于不能解决和延期解决的错误，不能由开发人员自己决定，一般要通过某种会议（评审会）通过才能认可。

- 测试人员查询状态为“FIXED”（已修复）的错误，验证错误是否已解决，做如下处理。
  - ✓ 问题解决了，置错误的状态为“VERIFIED”（已验证）。
  - ✓ 如问题没有解决，则置状态为“REOPENED”（重新打开）。
- 高级测试人员查询状态为“VERIFIED”（已验证）的缺陷，做如下处理。
  - ✓ 如果问题已经通过验证，并且没有错误，状态改为“CLOSED”（关闭）。
  - ✓ 如问题没有解决，则置状态为“REOPENED”（重新打开）。

**5. 错误流程管理原则**

错误流程管理遵照以下原则。

- 为了保证错误处理的正确性，需要有丰富测试经验的测试人员验证发现的错误是否是真正的错误，书写的测试步骤是否准确，可以重复。
- 每次对错误的处理都要保留处理信息，包括处理姓名、时间、处理方法、处理意见、Bug 状态。
- 拒绝或延期处理错误不能由程序员单方面决定，应该由项目经理、测试经理和设计经理共同决定。
- 错误修复后必须由报告错误的测试人员验证，确认已经修复后，才能关闭错误。
- 加强测试人员与程序员之间的交流，对于某些不能重复的错误，可以请测试人员补充详细的测试步骤和方法，以及必要的测试用例。

# 第 6 章　黑盒测试设计技术

**学习目标：**

1．了解什么是测试用例

2．熟悉测试用例设计的常用方法

3．了解其他测试经验

## 6.1　概述

本章介绍黑盒测试的概念和进行黑盒测试的目的与意义，及关于等价类划分、边界值分析、等测试用例设计方法的原理与实现，并从测试设计说明、测试用例说明、测试程序说明 3 个方面介绍如何编写测试用例。

## 6.2　测试用例设计方法

初涉软件测试者可能认为拿到软件后就可以立即进行测试，并希望马上找出软件的所有缺陷，这种想法就如同没有受过工程训练的开发工程师急于去编写代码一样。软件测试也是一个工程，也需要按照工程的角度去认识软件测试，在具体的测试实施之前，我们需要明白我们测试什么，怎么测试等，也就是说通过制定测试用例指导测试的实施。

### 6.2.1　什么是测试用例

所谓的测试用例就是将软件测试的行为活动，作一个科学化的组织归纳。软件测试是有组织性、步骤性和计划性的，而设计软件测试用例的目的，就是为了能将软件测试的行为转换为可管理的模式。软件测试是软件质量管理中最实际的行动，同时也是耗时最多的一项。基于时间因素的考虑，软件测试行为必须能够加以量化，才能进一步让管理阶层掌握所需要的测试过程，而测试用例就是将测试行为具体量化的方法之一。

简单地说，测试用例就是设计一个情况，软件程序在这种情况下，必须能够正常运行并且达到程序所设计的执行结果。如果程序在这种情况下不能正常运行，而且这种问题会重复发生，那就表示软件程序人员已经测出软件有缺陷，这时候就必须将这个问题标示出来，并且输入到问题跟踪系统内，通知软件开发人员。软件开发人员接获通知后，将这个问题修改完成于下一个测试版本内，软件测试工程师取得新的测试版本后，必须利用同一个用例来测试这个问题，确保该问题已修改完成。

因为不可能进行穷举测试，为了节省时间和资源、提高测试效率，必须要从数量极大的可用测试数据中精心挑选出具有代表性或特殊性的测试数据来进行测试。使用测试用例的好处主要体现在以下几个方面。

（1）在开始实施测试之前设计好测试用例，可以避免盲目测试并提高测试效率。

（2）测试用例的使用令软件测试的实施重点突出、目的明确。

（3）在软件版本更新后只需修正少部分的测试用例便可展开测试工作，降低工作强度、缩短项目周期。

（4）功能模块的通用化和复用化使软件易于开发，而测试用例的通用化和复用化则会使软件测试易于开展，并随着测试用例的不断精化其效率也不断攀升。

具体的黑盒测试用例设计方法包括等价类划分法、边界值分析法、错误推测法、因果图法、判定表驱动法、正交试验设计法等。应该说，这些方法是比较实用的，但采用什么方法，在使用时自然要针对开发项目的特点对方法加以适当的选择。下面我们讨论几种常用的方法。

### 6.2.2　如何编写测试用例

在前面我们谈了软件测试的一些基本概念，例如什么是软件测试，软件测试的基本要求，软件测试的基本思想。现在似乎到了我们要动手试试的时候了，如果你一听到要动手做测试，就捋起袖子敲键盘点鼠标，那就太急了。做软件测试有一个重要的步骤——编写软件测试用例。

什么是测试用例呢？测试用例其实就是一个个你测试的想法，你有了这些想法以后，详细地写下来，就成了测试用例。测试用例有几个重要的组成部分：

（1）简明扼要的标题。

（2）详细的步骤。

（3）正确的预期结果。

我们通过一个例子来说明。例如我们在测试记事本的时候，有了一个想法：应当测试一下这个软件能不能编辑中英文混合输入的内容，如图 6-1 所示。为了准确地实现我们想要测试的思想，我们要把它写下来，并且写下的内容要让任何人来看都没有歧义。

| |
|---|
| 测试用例：验证记事本成型可以编辑中英文混合的内容<br><br>测试步骤：<br>（1）运用记事本程序；<br>（2）切换到中文输入法，输入中文“学习编写”；<br>（3）切换到英文输入法，输入英文 Testcase；<br>（4）保存文件，文件名为 testcase.txt； |

（5）关闭记事本程序；
（6）双击 testcase.txt 以打开文件。

预期结果：
文件的内容是“学习编写 Testcase”。




图 6-1 测试记事本

测试用例还有一个优先级的概念，就是用来区分哪些用例更重要。一般可以分为 5 个级别，分别用 0～4 来表示，数字越小表示越重要。如果项目小，优先级的好处不容易显现出来。当项目比较大，时间又不宽裕时，可能只能执行更重要的测试用例，这个时候优先级的重要性就体现出来了。

大家也看到了，其实写测试用例并不难，但是它仍然容易出一些问题，例如：

（1）含混不清或者与内容不相符的标题。例如，上面的例子，如果用例叫“验证记事本可以编辑内容”，这个标题就没有准确表达出测试用例的实际内容。

（2）过于简单的步骤。这是一个容易犯的错误，很多朋友在编写用例的时候，总是写得很简单，例如上例中的多个步骤可能就会变成唯一的一步：“输入‘学习编写 Testcase’”，如果不是本人，其他人来看，肯定会引起歧义，怎么输入，是用键盘还是用复制的方法?那么写测试用例要详细到什么程度?就是让一个不了解你的工作的人来看，如果他的理解和你一样，说明你已经表达清楚了。

（3）没有写明预期结果。这是个严重的问题，如果没有预期的结果，那什么是对的什么又是错的呢？如果对错都分不清楚，做测试的意义又是什么呢？

（4）多个用例混在一个用例中。这也是刚入门的朋友容易出现的“好心办坏事”的情况，把测试用例写得特别长，包括了很多内容，这样很容易引起混淆，不如分开。而且，如果有多个用例混在一起，你的用例标题怎么写?另外，如果其中有几个用例通过，而另外几个没有通过，这时测试的结果很难记录，无论是把这个大的用例记录为通过或者不通过都不合适。

上面列出来的几个问题，大家可以尽量避免。实际上，写测试用例最难的地方是，如何把测试用例写得全面。这只能靠实践经验的积累了。你看完这节文章以后，可以拿记事本这个程序来练练，学着写几个测试用例，“看花容易绣花难”，所以要多试试。

### 6.2.3　测试用例的依据

有经验的测试人员通过他们长期从事的工作为我们这些没有经验的测试人员总结出了一些可供我们参考的依据。

（1）评审通过的需求规格说明书。

（2）评审通过的技术规格说明书。

（3）补充需求，隐含需求。

（4）用户体验及场景分析。

（5）基本成型的 UI。

（6）与产品、开发或用户沟通得到的系统关注点。

（7）产品功能、性能指标。

### 6.2.4　如何执行测试用例

虽然在前面我们讨论了如何编写软件测试用例，但如果你真是一位软件测试的入门者，你到单位报到后接手的第一项工作很可能是执行软件测试用例，而不是去编写。你不要因此而郁闷，这样的安排是合理的，因为你毕竟是个新手，执行软件测试用例是一个迅速熟悉当前测试工作的好机会，而且压力不大。因为在英语中执行测试用例是 run case，所以有些公司把执行测试用例叫做“跑 case”，想来也很形象。这也可以算是一种行话，你可以了解一下。

为方便讨论，我们以前面的测试用例为例。

| |
|---|
| 测试用例：验证记事本成型可以编辑中英文混合的内容<br><br>测试步骤：<br>（1）运用记事本程序；<br>（2）切换到中文输入法，输入中文“学习编写”；<br>（3）切换到英文输入法，输入英文 Testcase； |

（4）保存文件，文件名为 testcase.txt；
（5）关闭记事本程序；
（6）双击 testcase.txt 以打开文件。

预期结果：
文件的内容是“学习编写 Testcase”。

当我们面对这个用例的时候，我们首先要做的是清晰且正确地理解用例，不带半点含糊。测试的特点就是严谨，你来执行一个测试用例就是要贯彻用例编写者的测试思想，不能有误解或曲解，不能用自己的主观意志去代替原来的意思。例如，第一步“运行记事本程序”，你就应当清楚地知道“记事本”是哪个程序，如果有疑问马上问清楚，否则，如果真的把测试的产品都弄错了，一切就都白忙了，还浪费了时间。这个例子因为浅显，所以出现误解的可能性很小，而在实际的工作中，还是会有很多模棱两可的地方，这个时候我们不能偷懒，要勤学多问。

执行用例不能走样。例如，上例中的第二步，要求输入“学习编写”4 个字，如果你为了省事，复制了这几个字，每次都是粘贴过来，快是快了，却违背了“原著”的意思，这样是不可以的。用例编写者要求用输入法来输入，肯定是有道理的。如果你发现没有检测“粘贴”的测试用例，可以建议增加，但不能在执行的时候就偏离了用例的本意。说一件万一的事儿，如果这个软件通过了你的测试，发布给用户，用户却发现不能输入，只能粘贴，这个责任你能负得起吗？

大家可能都知道，做软件测试要细心，这个要求在执行用例的过程中表现得很明显。我们在执行一个测试用例的时候，不但要注意实际结果是否与预期结果是一致的，而且在整个过程中都要保持观察。例如上例中，如果第四步执行保存后，你发现文件名并不是自己输入的 testcase.txt，这时你就应当停下来，因为这就是 Bug。

我们执行测试用例的目的是什么？就是发现 Bug，所以，我们在执行测试用例的过程中，要收集好发现的问题，不能有遗漏。在实际工作中，执行测试用例的过程一般都是紧张的，工作量很大，并不像我们今天在这里讨论的这么轻松，因为你要不停地往前赶，所以容易出现一些遗漏的问题。

每当发现一个问题，我们都要做好记录，而不要总以为自己能记得住，好记性不如烂笔头。Bug 是最能证明测试工程师工作成绩的东西，好不容易发现了，如果还被自己遗漏了，岂不令人懊悔？而且，还给产品留下了一个隐患。

前面说过，执行测试用例是一个很好的学习机会。你可以在工作之余，去体会测试用例编写者的测试思想，而测试思想对于测试工程师来说是最重要的。你可以想一想，哪些测试用例是自己没有想到的？测试用例编写者的思维主线是什么？经过这样的琢磨，你对测试工作就会有进一步的认识和体会。你还可以尝试着去扩充测试用例，这是

一个锻炼和提高自己测试能力的好方法。

## 6.3　测试用例设计方法

### 6.3.1　等价类划分法

等价类划分法是一种典型的黑盒测试方法，用这一方法设计测试用例完全不考虑程序的内部结构，只根据对程序的要求和说明，即需求规格说明书。我们必须仔细分析和推敲说明书的各项需求，特别是功能需求。把说明书中对输入的要求和输出的要求区别开来并加以分解。

由于穷举测试工作量太大，以至于无法实际完成，促使我们在大量的可能数据中选取其中的一部分作为测试用例。例如，在不了解等价分配技术的前提下，我们做计算器程序的加法测试时，测试了 1+1，1+2，1+3 和 1+4 之后，还有必要测试 1+5 和 1+6 吗，能否放心地认为它们是正确的？我们感觉 1+5 和 1+6，与前面的 1+1，1+2 都是很类似的简单加法。

等价类划分的办法是把程序的输入域划分成若干部分，然后从每个部分中选取少数代表性数据作为测试用例。每一类的代表性数据在测试中的作用等价于这一类中的其他值，也就是说，如果某一类中的一个例子发现了错误，这一等价类中的其他例子也能发现同样的错误；反之，如果某一类中的一个例子没有发现错误，则这一类中的其他例子也不会查出错误（除非等价类中的某些例子属于另一等价类，因为几个等价类是可能相交的）。使用这一方法设计测试用例，首先必须在分析需求规格说明的基础上划分等价类，列出等价类表。

**1．划分等价类和列出等价类表**

等价类是指某个输入域的子集合。在该子集合中，各个输入数据对于揭露程序中的错误都是等效的。并合理地假定：测试某等价类的代表值就等于对这一类其他值的测试。

因此，可以把全部输入数据合理地划分为若干等价类，在每一个等价类中取一个数据作为测试的输入条件，就可以用少量代表性的测试数据取得较好的测试结果。等价类划分有两种不同的情况：有效等价类和无效等价类。

有效等价类：指对于程序的规格说明来说是合理的、有意义的输入数据构成的集合。利用有效等价类可检验程序是否实现了规格说明中所规定的功能和性能。

无效等价类：与有效等价类的定义恰巧相反。

设计测试用例时，要同时考虑这两种等价类。因为软件不仅要能接收合理的数据，也要能经受意外的考验。这样的测试才能确保软件具有更高的可靠性。

下面给出 6 条确定等价类的原则。

（1）在输入条件规定了取值范围或值的个数的情况下，可以确立一个有效等价类和

两个无效等价类。

（2）在输入条件规定了输入值的集合或者规定了“必须如何”的条件的情况下，可以确立一个有效等价类和一个无效等价类。

（3）在输入条件是一个布尔量的情况下，可确定一个有效等价类和一个无效等价类。

（4）在规定了输入数据的一组值（假定 $n$ 个），并且程序要对每一个输入值分别处理的情况下，可确立 $n$ 个有效等价类和一个无效等价类。

（5）在规定了输入数据必须遵守的规则的情况下，可确立一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）。

（6）在确知已划分的等价类中，各元素在程序处理中的方式不同的情况下，则应再将该等价类进一步地划分为更小的等价类。

在确立了等价类之后，建立等价类表，列出所有划分出的等价类如表 6-1 所示。

**表 6-1 等价类表示例**

| 输入条件 | 有效等价类 | 无效等价类 | 输入条件 | 有效等价类 | 无效等价类 |
|---|---|---|---|---|---|
| … | … | … | … | … | … |

**2．确定测试用例**

根据已列出的等价类表，按以下步骤确定测试用例：

（1）为每个等价类规定一个唯一的编号。

（2）设计一个新的测试用例，使其尽可能多地覆盖尚未覆盖的有效等价类。重复这一步，最后使得所有有效等价类均被测试用例所覆盖。

（3）设计一个新的测试用例，使其只覆盖一个无效等价类。重复这一步使所有无效等价类均被覆盖。

在寻找等价区间时，想办法把软件的相似输入、输出、操作分成组。这些组就是等价区间。请看一些例子。

在两数相加用例中，测试 1+13 和 1+99 999 999 似乎有点不同。这是一种直觉，一个是普通加法，而另一个似乎有些特殊，这种直觉是对的。程序对 1 和最大数值相加的处理与对两个小一些的数值相加的处理有所不同。后者必须处理溢出情况。因为软件操作可能不同，所以这两个用例属于不同的等价区间。

如果具有编程经验，就可能会想到更多可能导致软件操作不同的“特殊”数值。如果不是程序员，也不用担心，你很快就会学到这种技术，无须了解代码细节就可以运用。图 6-2 所示是复制的多种方法，给出了选中编辑菜单后显示复制和粘贴命令的计算器程序。每一项功能（即复制和粘贴）有 5 种执行方式。要想复制，可以单击复制菜单命令，按 C 键，按 Ctrl+C 或 Ctrl+Shift+C 组合键。任何一种输入途径都会把当前数值复制到剪贴板中，一一执行同样的输出操作，产生同样的结果。

如果要测试复制命令，可以把这 5 种输入途径划分减为 3 个，单击菜单命令，按 C 键和按 Ctrl+C 组合键。对软件质量有了信心之后，知道无论以何种方式激活复制功能都工作正常，甚至可以进一步缩减为 1 个区间，例如按 Ctrl+C 组合键。




图 6-2　复制的多种方法

再看下一个例子。看一下在标准的另存为对话框（如图 6-3 所示）中输入文件名称的情形。Windows 文件名可以包含除了“/”、“\”、“:”、“*”、“?”、“"”、“<”、“>”和“|”之外的任意字符。文件名长度是 1～255 个字符。如果为文件名创建测试用例，等价区间有合法字符、非法字符、合法长度的名称、过长名称和过短名称。

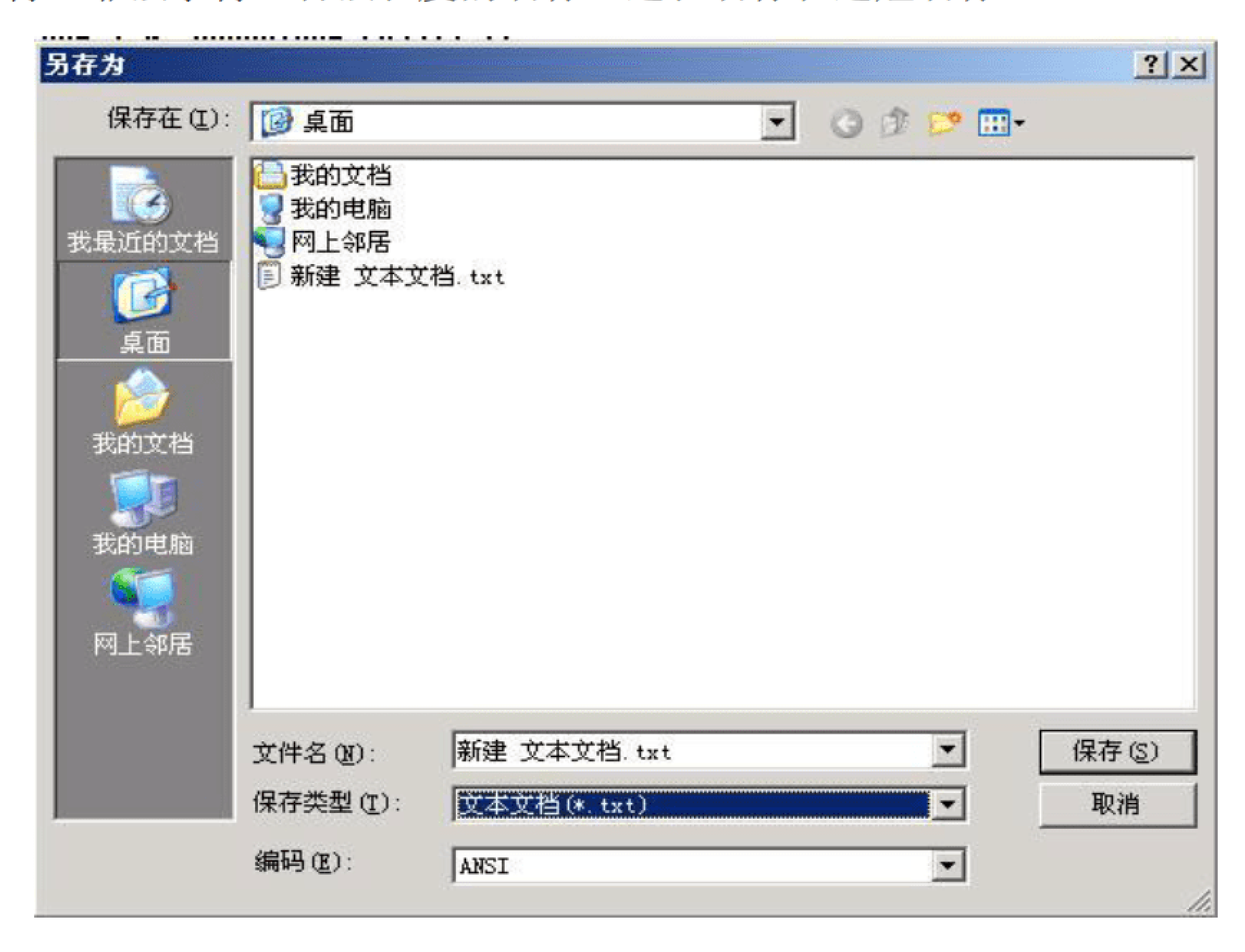



图 6-3　另存为对话框

例题：根据下面给出的规格说明，利用等价类划分的方法，给出足够的测试用例。“一个程序读入 3 个整数，把这 3 个数值看作一个三角形的 3 条边的长度值。这个程序要打印出信息，说明这个三角形是不等边的，是等腰的，还是等边的。”

我们可以设三角形的 3 条边分别为 A，B，C。如果它们能够构成三角形的 3 条边，必须满足：A>0，B>0，C>0，且 A+B>C，B+C>A，A+C>B。

如果是等腰的，还要判断 A=B，或 B=C，或 A=C。

如果是等边的，则需判断是否 A=B，且 B=C，且 A=C。列出等价类表，如表 6-2 所示。

**表 6-2　等价类表**

| 输 入 条 件 | 有效等价类 | 无效等价类 |
|---|---|---|
| 是否三角形的 3 条边 | （A>0），（1）<br>（B>0），（2）<br>（C>0），（3）<br>（A+B>C），（4）<br>（B+C>A），（5）<br>（A+C>B），（6） | （A≤0），（7）<br>（B≤0），（8）<br>（C≤0），（9）<br>（A+B≤C），（10）<br>（B+C≤A），（11）<br>（A+C≤B），（12） |
| 是否等腰三角形 | （A=B），（13）<br>（B=C），（14）<br>（C=A），（15） | （A≠B）and（B≠C）and（C≠A），（16） |
| 是否等边三角形 | （A=B）and（B=C）and（C=A），（17） | （A≠B），（18）<br>（B≠C），（19）<br>（C≠A），（20） |

设计测试用例：输入顺序是 A，B，C，如表 6-3 所示。

**表 6-3　测试用例**

| 序号 | 【A，B，C】 | 覆盖等价类 | 输　出 |
|---|---|---|---|
| 1 | 【3,4,5】 | （1），（2），（3），（4），（5），（6） | 一般三角形 |
| 2 | 【0,1,2】 | （7） | 不能构成三角形 |
| 3 | 【1,0,2】 | （8） | |
| 4 | 【1,2,0】 | （9） | |
| 5 | 【1,2,3】 | （10） | |
| 6 | 【1,3,2】 | （11） | |
| 7 | 【3,1,2】 | （12） | |
| 8 | 【3,3,4】 | （1），（2），（3），（4），（5），（6），（13） | 等腰三角形 |
| 9 | 【3,4,4】 | （1），（2），（3），（4），（5），（6），（14） | |
| 10 | 【3,4,3】 | （1），（2），（3），（4），（5），（6），（15） | |
| 11 | 【3,4,5】 | （1），（2），（3），（4），（5），（6），（16） | 非等腰三角形 |

续表

| 序号 | 【A，B，C】 | 覆盖等价类 | 输　　出 |
|---|---|---|---|
| 12 | 【3,3,3】 | （1），（2），（3），（4），（5），（6），（17） | 是等边三角形 |
| 13 | 【3,4,4】 | （1），（2），（3），（4），（5），（6），（14），（18） | 非等边三角形 |
| 14 | 【3,4,3】 | （1），（2），（3），（4），（5），（6），（15），（19） | |
| 15 | 【3,3,4】 | （1），（2），（3），（4），（5），（6），（13），（20） | |

请记住，等价分配的目标是把可能的测试用例组合缩减到仍然足以满足软件测试需求为止。因为，选择了不完全测试，就要冒一定的风险，所以必须仔细选择分类。

关于等价分配最后要讲的一点是，这样做有可能不客观。科学有时也是一门艺术。测试同一个复杂程序的两个软件测试员，可能会制定出两组不同的等价区间。只要审查等价区间的人都认为它们足以覆盖测试对象就可以了。

### 6.3.2　边界值分析法

人们从长期的测试工作经验得知，大量的错误是发生在输入或输出范围的边界上的，而不是在输入范围的内部。因此针对各种边界情况设计测试用例，可以查出更多的错误。例如，在做三角形计算时，要输入三角形的 3 个边长 A，B 和 C。这 3 个数值应当满足 A>0、B>0、C>0、A+B>C、A+C>B、B+C>A，才能构成三角形。但如果把 6 个不等式中的任何一个大于号“>”错写成大于等于号“≥”，那就不能构成三角形。问题恰恰出现在容易被疏忽的边界附近。这里所说的边界是指相当于输入等价类和输出等价类而言，稍高于其边界值及稍低于其边界值的一些特定情况。

**1．边界条件**

我们可以想象一下，如果在悬崖峭壁边可以自信地安全行走，平地就不在话下了。如果软件在能力达到极限时能够运行，那么在正常情况下一般也就不会有什么问题。

边界条件是特殊情况，因为编程从根本上说不怀疑边界有问题。奇怪的是，程序在处理大量中间数值时都是对的，但是可能在边界处出现错误。下面的一段源代码说明了在一个极简单的程序中是如何产生边界条件问题的。

```
rem create a 10 element integer array
rem initialize each element to-1
dim data (10) as integer
dim i as integer
for i=1 to 10 data(i)=-1
next i
end
```

这段代码的意图是创建包含 10 个元素的数组，并为数组中的每一个元素赋初值–1。看起来相当简单。它建立了包含 10 个整数的数组 data 和一个计数值 i。For 循环是从 1～

10，数组中从第 1 个元素到第 10 个元素被赋予数值–1。那么边界问题在哪儿呢?

在大多数开发语言脚本中，应当以声明的范围定义数组，在本例中定义语句是 dim data（10）as interger，第一个创建的元素是 data（0），而不是 data（1）。该程序实际上创建了一个从 data（0）～data（10）共 11 个元素的数组。程序从 1～10 循环将数组元素的值初始化为–1，但是由于数组的第一个元素是 data（0），因此它没有被初始化。程序执行完毕，数组值如下：

```
data（0）=0           data（6）=-1
data（1）=-1          data（7）=-1
data（2）=-1          data（8）=-1
data（3）=-1          data（9）=-1
data（4）=-1          data（10）=-1
data（5）=-1
```

注意 data（0）的值是 0，而不是–1。如果这位程序员以后忘记了，或者其他程序员不知道这个数据数组是如何初始化的，那么他就可能会用到数组的第 1 个元素 data（0），以为它的值是–1。诸如此类的问题很常见，在复杂的大型软件中，可能导致极其严重的软件缺陷。

**2．次边界条件**

上面讨论的普通边界条件是最容易找到的。它们在产品说明书中有定义，或者在使用软件的过程中确定。而有些边界在软件内部，最终用户几乎看不到，但是软件测试仍有必要检查。这样的边界条件称为次边界条件或者内部边界条件。

寻找这样的边界不要求软件测试员具有程序员那样阅读源代码的能力，但是要求大体了解软件的工作方式。2 的乘方和 ASCII 表就是这样的例子。

（1）2 的乘方。

计算机和软件的计数基础是二进制数，用位（Bit）来表示 0 和 1，一个字节（Byte）由 8 位组成，一个字（Word）由两个字节组成等。表 6-4 中列出了常用的 2 的乘方单位及其范围或值。

**表 6-4　软件中 2 的乘方**

| 术　语 | 范 围 或 值 | 术　语 | 范 围 或 值 |
|---|---|---|---|
| 位 | 0 或 1 | 千 | 1 024 |
| 双位 | 0～15 | 兆 | 1 048 576 |
| 字节 | 0～255 | 亿 | 1 073 741 824 |
| 字 | 0～65 535 | 万亿 | 1 099 511 627 776 |

表 6-4 中所列的范围和值是作为边界条件的重要数据。除非软件向用户提出这些范

围，否则在需求文档中不会指明。然而，它们通常由软件内部使用，外部是看不见的，当然，在产生软件缺陷的情况下可能会看到。

在建立等价区间时，要考虑是否需要包含 2 的乘方边界条件。例如，如果软件接受用户输入 1～1000 范围内的数字，谁都知道在合法区间中包含 1 和 1000，也许还要有 2 和 999。为了覆盖任何可能的 2 的乘方次边界，还要包含临近双位边界的 14，15 和 16，以及临近字节边界的 254，255 和 256。

（2）ASCII 表。

另一个常见的次边界条件是 ASCII 字符表。表 6-5 所示是部分 ASCII 值表的清单。

**表 6-5　部分 ASCII 值表**

| 字　符 | ASCII 值 | 字　符 | ASCII 值 | 字　符 | ASCII 值 | 字　符 | ASCII 值 |
|---|---|---|---|---|---|---|---|
| Null | 0 | B | 66 | 2 | 50 | a | 97 |
| Space | 32 | Y | 89 | 9 | 57 | b | 98 |
| / | 47 | Z | 90 | : | 58 | y | 121 |
| 0 | 48 | [ | 91 | @ | 64 | Z | 122 |
| 1 | 49 | ' | 96 | A | 65 | { | 123 |

注意，表 6-5 不是结构良好的连续表。0～9 的后面 ASCII 值是 48～57。斜杠字符（/）在数字 0 的前面，而冒号字符“:”在数字 9 的后面。大写字母 A～Z 对应 65～90。小写字母对应 97～122。这些情况都代表次边界条件。

如果测试进行文本输入或文本转换的软件，在定义数据区间包含哪些值时，参考一下 ASCII 表是相当明智的。例如，如果测试的文本框只接受用户输入字符 A～Z 和 a～z，就应该在非法区间中包含 ASCII 表中这些字符前后的值@、[、和{。

（3）其他一些边界条件。

另一种看起来很明显的软件缺陷来源是当软件要求输入时（比如在文本框中），不是没有输入正确的信息，而是根本没有输入任何内容，只按了 Enter 键。这种情况在产品说明书中常常被忽视，程序员也可能经常遗忘，但是在实际使用中却时有发生。程序员总会习惯性地认为用户要么输入信息，不管是看起来合法的或非法的信息，要么就会选择 Cancel 键放弃输入，如果没有对空值进行好的处理的话，恐怕程序员自己都不知道程序会引向何方。

正确的软件通常应该将输入内容默认为合法边界内的最小值，或者合法区间内的某个合理值，否则，返回错误提示信息。

因为这些值通常在软件中进行特殊处理，所以不要把它们与合法情况和非法情况混在一起，而要建立单独的等价区间。

**3．边界值的选择方法**

边界值分析是一种补充等价划分的测试用例设计技术，它不是选择等价类的任意元

素，而是选择等价类边界的测试用例。实践证明，为检验边界附近的处理而专门设计测试用例，常常取得良好的测试效果。边界值分析法不仅重视输入条件边界，而且也适用于输出域测试用例。

对边界值设计测试用例，应遵循以下几条原则：

① 如果输入条件规定了值的范围，则应取刚达到这个范围的边界的值，以及刚刚超越这个范围边界的值作为测试输入数据。

② 如果输入条件规定了值的个数，则用最大个数、最小个数、比最小个数少 1、比最大个数多 1 的数作为测试数据。

③ 如果输出条件规定了值的范围，则应取刚达到这个范围的边界的值，以及刚刚超越这个范围边界的值作为测试输入数据。

④ 如果输出条件规定了值的范围，则应取刚达到这个范围的边界的值，以及刚刚超越这个范围边界的值作为测试输入数据。

⑤ 如果程序的规格说明给出的输入域或输出域是有序集合，则应选取集合的第一个元素和最后一个元素作为测试用例。

⑥ 如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构边界上的值作为测试用例。

⑦ 分析规格说明，找出其他可能的边界条件。

### 6.3.3 错误推测法

错误推测法就是基于经验和直觉推测程序中所有可能存在的错误，有针对性地设计测试用例的方法。

错误推测法的基本思想是列举出程序中所有可能有的错误和容易发生错误的特殊情况，根据它们选择测试用例。例如，设计一些非法、错误、不正确和垃圾数据进行输入测试是很有意义的。如果软件要求输入数字，就输入字母。如果软件只接受正数，就输入负数。如果软件对时间敏感，就看它在公元 3000 年是否还能正常工作。还有，例如，在单元测试时曾列出的许多在模块中常见的错误，以前产品测试中曾经发现的错误等，这些就是经验的总结。另外，输入数据和输出数据为 0 的情况，或者输入表格为空格或输入表格只有一行，这些都是容易发生错误的情况。可选择这些情况下的例子作为测试用例。

## 6.4 其他测试经验

余下的黑盒子测试技术不像已经描述的各种数据测试和状态测试那样独立。如果对整个程序数据进行了等价分配，创建了详细的状态图，并且开发完成了相关的测试案例，就会找出用户可能发现的大多数软件缺陷。

剩下的是发现衍生问题的技术。假如软件缺陷是活生生的寄生虫，就应该有自己的想法和行为方式。这样寻找也许有一点主观，没有实实在在的理由，但是如果要找出所有的软件缺陷，就必须有一点创造力。

### 6.4.1　像愚笨的用户那样做

为了礼貌一些，正确的说法也许应该是无经验的用户或新用户，但是事实上都是一回事。一个不熟悉软件的人面对程序时，他会做出令人永远想不到的举动。他们会输入程序员无从想象的数据。他们会在中途变卦，退回去执行其他操作。他们冲浪遇到某个站点，可能会单击不应该单击的东西。他们会发现开发小组完全遗漏的软件缺陷。

软件测试员看到一个没有任何测试经验的人只花 5 分钟来使用软件并使其崩溃，一定会感到沮丧吧？他们是怎样做的?他们不遵循任何规则，也不做任何假定。

在设计测试案例或者初次查看软件时，要设法像愚笨的用户那样想问题。抛开关于软件应该如何工作的先入之见。如果可能，找一个其他专业的朋友来整理思路。假设他什么也不会。把这些测试案例加入到已经设计好的测试案例库中，就会更加全面。

### 6.4.2　在已经找到软件缺陷的地方再找找

在已经找到软件缺陷的地方再找的原因有两个：

（1）找到的软件缺陷越多，就说明那里的软件缺陷越多。如果发现在不同的特性中找出了大量上边界条件软件缺陷，那么明智的做法是对所有特性着重测试上边界条件。

（2）许多程序员倾向于只修复报告出来的软件缺陷，不多也不少。如果报告软件缺陷是启动—终止—再启动 255 次导致冲突，程序员就只修复这个问题。也许是内存泄漏导致这个问题出现，程序员找到症结并将其修复。当拿回软件重新测试时，一定要重新执行同样的测试 256 次以上。在这个范围之外极有可能存在其他的内存泄漏问题。

### 6.4.3　凭借经验、直觉和预感

要想成为真正的软件测试员，积累经验是不可替代的。没有更好的学习工具，也没有比客户第一次打电话报告刚经过测试的软件中存在缺陷一事更好的教训了。

经验和直觉是不可言传的，必须经过长期的积累。运用现在学到的全部技术进行测试，仍然有可能遗漏重要的软件缺陷。这是无法更改的事实。随着在职业生涯中逐步提高，学习测试不同类型和规模的产品，就会得到各种提示和技巧以便更加有效地找出软件缺陷。重新开始测试新软件，就可以很快找出以前会遗漏的软件缺陷。

记录哪些技术有效，哪些不行。尝试不同的途径。如果认为有可疑之处，就要仔细探究。按照预感行事，直至证实这是错误为止。

# 第 7 章　系 统 测 试

**学习目标：**

1．了解什么是系统测试

2．熟悉系统测试的方法

## 7.1　系统测试概念

在测试的四个级别中，系统测试是最重要的一个测试阶段。系统测试的任务除了要证明被测系统的功能和结构的稳定性外，还要有一些非功能测试，比如用户界面，兼容性测试，安装性测试等，最终目的是为了确保软件产品能够被用户或操作者接受。

在实际软件项目的开发中，系统测试常常不是十分正式，测试的主要目标不再是找出缺陷，而是确认其功能和性能。很多软件组织，尤其是中小型软件组织经常在产品交付日期截止之前压缩系统测试的时间，这种做法是不正确的。应该是把系统测试看成是产品提交给用户之前的最后一道防线，给予足够的重视。

### 7.1.1　什么是系统测试

由于软件只是计算机系统的一个组成部分，软件开发完成后，还要与系统的其他部分（如计算机硬件及相关的外围设备、数据收集和传输机构、操作系统、Web 服务器、数据库服务器等）结合起来才能运行。所以在整个系统投入运行前，要对系统的各部分进行集成。也就是说，在各个部分都能够正常运行的前提下，确保在实际运行的软、硬件环境下也能够相互配合，正常工作。

一般而言，系统测试就是将已经集成好的软件系统，作为整个计算机系统的一个元素，与计算机硬件、某些支持软件、数据和人员等其他系统元素结合在一起，在实际运行（使用）环境下，对计算机系统进行一系列的集成测试。实际上，就是对被测系统的各个组成部分进行综合检验。虽然系统测试的类型有很多，而且每一种测试都有特定的目标，但所有的测试工作都是为了验证已经集成的系统中的每个部分都可以正确地完成指定的功能。

系统测试属于黑盒测试范畴，不再对软件的源代码进行分析和测试。

系统测试的目标在于通过与系统的需求规格说明进行比较，检查软件是否存在与系统规格不符合或与之矛盾的地方，以验证软件系统的功能和性能等满足其规格说明所指定的要求。因此，测试设计人员应该主要根据需求分析说明书来设计系统测试的测试

用例。

### 7.1.2　系统测试的组织和分工

系统测试应该由测试组组长组织，测试分析员负责设计和实现测试脚本和测试用例，测试员负责执行测试脚本中记录的测试用例。在一些小型测试组中，测试分析员和测试员可以由同一个人来担任。系统测试还要有独立测试观察员监控测试过程，同时也可以找一个客户代表非正式地观看测试过程。

邀请客户代表参与系统测试的好处有很多，可以与客户建立一个良好的平台，显示被测系统的运行情况和面貌，也可以得到一部分反馈意见。

测试组组长要与负责管理 IT 设备的人员联系搭建好系统测试的软、硬件平台。然后，由测试组组长制订软件测试计划，此过程中需要与开发人员多多沟通。完成系统测试后，需要提交系统测试输出的大量的文档，包括测试结果记录表格、系统测试日志和全面的系统测试总结报告。

### 7.1.3　系统测试分析

在系统测试的各个环节中，比较关键和困难的是系统测试用例的设计阶段。测试人员在做系统测试分析时，不妨分别从用户层、应用层、功能层、子系统层、协议层等几个层次入手。

**1．用户层**

因为用户层面向的是产品最终的使用者——用户，因此用户层的测试主要围绕着诸如用户界面的规范性、友好性、可操作性，系统对用户的支持，以及数据的安全性等方面展开。测试的对象应该有用户手册、使用帮助以及支持客户的其他产品技术手册，检查其是否正确、是否易于理解。是否人性化。另外，在确保用户界面能够通过测试对象控件或入口得到相应访问的情况下，还应该测试用户界面的风格是否满足用户要求，例如界面是否美观、直观、友好，是否更加人性化。

对于用户层的测试还应该注意可维护性测试和安全性测试。

可维护性是指实施系统软、硬件维护的容易性，降低维护工作对系统正常运行带来的影响。安全性主要包括两部分：数据安全性和操作安全性。系统可以访问的数据必须符合规定，系统可以执行的操作权限也必须符合规定。

**2．应用层**

应用层的测试主要是针对产品工程应用或行业应用的测试。从应用软件系统的角度出发，模拟实际应用环境，对系统的兼容性、可靠性、性能等进行测试。针对整个系统的应用层测试，包含并发性能测试、负载测试、压力测试、强度测试、破坏性测试。

**3．功能层**

功能层的测试是要检测系统是否已经实现需求规格说明中定义的功能，以及系统功

能之间是否存在类似共享资源访问冲突的情况。

**4. 子系统层**

子系统层的测试是针对产品内部结构性能的测试。关注子系统内部的性能、子系统之间接口的瓶颈。如果只有一个单个子系统，就要关注整个系统各种软硬件、接口配合情况下的整体性能。

**5. 协议/指标层**

协议/指标层测试是针对系统所支持的协议，进行协议一致性测试和协议互通测试。

### 7.1.4 系统测试环境

不同（版本）的操作系统、不同（版本）的数据库、不同（版本）的网络服务器、应用服务器，再加上不同的系统架构等的组合，使得要构建的软件测试环境多种多样；而且，随着软件运行环境的多样性、配置各种相关参数的浩繁和测试软件的兼容性等方面的需要，使得构建软件测试环境的工作变得更为复杂和频繁。因此，软件测试环境的搭建是软件测试实施的一个重要阶段和环节。

在软件开发过程中，创建可复用的软件构件库，对于提高开发质量、减少开发费用、保证开发进度有极重要的辅助作用。同样地，测试人员也可以通过构建软件测试环境库的方式来实现软件测试环境的复用，节省宝贵的测试时间。

软件测试环境库要存放在单独的硬盘分区上，不要和其他经常需要读写的文件放在一起，并尽量不要对软件测试环境库所在的硬盘分区进行磁盘整理，以免对镜像文件造成破坏。此外，软件测试环境库存放在网络文件服务器上安全性太低，最好将它们制作成可自启动的光盘，由专人进行统一管理；一旦需要搭建测试环境时，就可通过网络、自启动的光盘或硬盘等方式，由专人负责将镜像文件恢复到指定的目录中去。这项工作一旦完成后，被还原的硬盘上的原有信息将完全丢失，所以请慎重使用，可先把硬盘上原有的重要的文件资料提前备份，以防不测。

## 7.2 系统测试的方法

系统测试的方法非常多，下面将对一些比较常见的系统测试方法做概要性的介绍。

### 7.2.1 功能测试

**1. 基本概念**

功能测试（Functional Test）属于黑盒测试，是系统测试中最基本的测试。它不用考虑软件内部的具体实现过程，主要是根据产品的需求规格说明和测试需求列表，验证产品是否符合需求规格说明。功能测试主要是为了发现以下几类错误：

（1）是否有不正确或遗漏了的功能?

（2）功能实现是否满足用户需求和系统设计的隐式需求?

（3）能否正确地接受输入?能否正确地输出结果?

功能测试要求测试人员对被测系统的需求规格说明、业务功能都非常熟悉，同时掌握一定的测试用例设计方法。除此之外，测试人员还需要了解相关的业务知识，要对测试过程中的细节问题有所理解。只有达到了这样的要求，测试人员才能够设计出好的测试方案和测试用例，高效地进行功能测试。

**2. 分析方法**

需求规格说明是功能测试的基本输入。因此在做功能测试前，首先应该对需求规格说明进行分析，明确功能测试的重点。对需求规格说明的分析可分为以下几个步骤：

（1）对所有的功能需求（包括隐含的功能需求）加以标识。

（2）对所有可能出现的功能异常进行分类分析并加以标识。

（3）对所有标识的功能需求确定优先级。可以根据功能测试工作量的大小，以及特定的约束指标（如风险等级）来决定对每个功能投入多少测试资源。通常按照应用要求的不同把软件功能分为关键功能和非关键功能。其中关键功能是指系统的主要功能，即用户工作时必须使用的功能。比如在画图工具软件中，对图形的显示就是关键功能。非关键功能是在整个系统中起到辅助作用的功能，如界面设计得是否美观、是否友好等。

（4）对每个功能进行测试分析，分析其是否可测，采用何种测试方法，测试的入口条件，可能的输入，预期输出等。

（5）确定是否需要开发测试脚本或借助工具录制脚本。

（6）确定要对哪些测试使用自动化测试，对哪些测试使用手工测试。

除了要选取合适的测试技术外，在书写功能测试用例时也应该遵循一定的规范，这样便于功能测试过程甚至整个软件测试过程的监控和管理，提高测试工作的效率。

例如，有关“用户界面”的功能测试项目如下：

- 页面链接检查。每一个链接是否都有对应的页面，并且页面之间切换是否正确。
- 相关性检查。删除 / 增加一项会不会对其他项产生影响?如果产生影响，这些影响是否都正确。
- 按钮功能检查。检查如 Update、Cancel、Delete、Save 等功能是否正确。
- 字符串长度检查。输入超出需求规格要求的字符串长度的内容，看系统是否检查字符串长度，会不会出错。
- 字符类型检查。在应该输入指定类型的内容的地方输入其他类型的内容。例如，在本应该输入整型的地方输入字符类型，看系统是否检查输入的类型，是否报错。
- 标点符号检查。输入内容包括各种标点符号，特别是空格、各种引号、回车键，看系统处理是否正确。
- 中文字符处理。在可以输入中文的系统输入中文，看是否会出现乱码或出错。

- 检查带出信息的完整性。在查看信息和更新信息时，查看所填写的信息是不是全部带出，带出信息和所添加的信息是否一致。
- 信息重复。在一些需要命名，且名字应该唯一的信息中输入重复的名字或 ID，看系统是否报错，重名情况还要考虑是否区分大小写，以及在输入内容的前后输入的空格是否滤掉等，检查系统是否做出正确处理。
- 删除功能检查。在一些可以一次删除多个信息的地方，不选择任何信息，单击 delete 按钮，看系统如何处理，是否出错；然后选择一个和多个信息，进行删除，看是否正确处理。
- 检查添加和修改是否一致。检查添加和修改信息的要求是否一致，例如添加要求必填的项，修改也应该必填；添加规定为整型的项，修改也必须为整型。
- 修改重名检查。修改时把不能重名的项改为已有项的名字，看是否处理并报错。同时，也要注意报错信息是否正确。
- 重复提交表单。一条已经成功提交的记录，后退（Back）后再提交，看系统是否做了处理，以及处理是否正确。
- 检查多次使用后退（Back）键的情况。在可以后退的地方，选择后退，回到先前的页面，再后退，重复多次，看是否会出错。
- 搜索检查。在有搜索功能的地方输入系统中存在和不存在的内容，看搜索结果是否正确。如果可以输入多个搜索条件，可以同时添加合理和不合理的条件，看系统处理是否正确。
- 输入信息位置。注意在光标停留处输入信息时，光标和所输入的信息是否跳到别处。
- 上传/下载文件检查。上传/下载文件的功能是否实现，上传文件是否能打开，对上传文件的格式有何规定，系统是否有解释信息，并检查系统是否能够做到。
- 必填项检查。应该填写的项没有填写时系统是否都做了处理，对必填项是否有提示信息，如在必填项前加“*”。
- 快捷键检查。是否支持常用快捷键，如 Ctrl+C、Ctrl+V、Backspace 等，对一些不允许输入信息的字段，对快捷方式是否也做了限制。
- 回车键检查。在输入结束后直接按回车键，看系统处理如何，是否报错。

**3．测试用例设计**

功能测试用例是功能测试工作的核心，常见的测试用例设计方法有如下几种：等价类测试；边界值测试、因果图测试；基于判定表的测试；正交实验设计法；错误猜测法。

**4．功能测试覆盖**

功能覆盖中最常见的是需求覆盖，其含义是通过设计一定的测试用例，要求覆盖各项需求。

例如，如果已经确定了所有性能需求，则可以引用性能测试结果来评测，最后得到

75%的性能需求已被覆盖。需求覆盖率的计算公式为：

需求覆盖率=被验证到的需求数量/总的需求数量

另外，还有一种覆盖称为接口覆盖，又称为入口点覆盖。要求设计一定的测试用例，使得系统的每一个接口都被覆盖到。

### 7.2.2　功能易用性测试

功能易用性的概念范围很广，这里列出了一些比较重要的功能易用性测试项，如下所示。

（1）业务符合性：软件使用的目的是替代部分人工劳动，提高工作效率，因此，软件必须符合其所服务的领域的业务逻辑。这就要求软件的界面风格、表格设计、业务流程、数据加密机制等的设计必须符合相关的法律、法规、业界标准规范以及使用人员的习惯。

（2）功能定制性：为了适应用户需求的不断变化，软件功能应当能够灵活定制，如电子政务软件的公文流转节点，应可以灵活定义；工资软件中部门结构和人员归属应可灵活调整等。

（3）业务模块的集成度：在一个系统中业务模块之间有可能存在较紧密的关联，例如在 ERP 系统中，采购某些零部件之后必须进行质检，这样的业务需求造成“采购管理”模块与“质量检测”模块存在直接的关联，那么用户能否在“采购管理”用户操作界面下，直接进入“质量检测”模块，并且“采购管理”模块中的零部件数据能否直接传递给“质量检测”模块。

（4）数据共享能力：“一次输入、多处应用”不仅能够减少用户的重复输入工作，更有效地保证了数据的正确性。在软件设计中必须充分考虑数据库表的关联和数据重用问题，最大程度地减少用户的重复输入，同时保证数据传递的一致性。

（5）约束性：对于流程性比较强的业务操作，上一步操作完成之后，要强制进行下一步操作，这时需要软件以向导或与屏蔽无关操作的方式来限制用户的操作。另外，应以屏蔽或提示的方式阻止用户输入非法字符或进行损害数据和系统的操作，这样才能有效地避免用户犯错误，同时也减少了系统出现异常的概率，提高系统的安全可靠性。

（6）交互性：包括用户操作的可见性和系统对用户的反馈。对于用户的每一步操作都应有所回应或者提示，使用户清晰地看到系统的运行状态。例如，在执行复制操作时，至少应该向用户反馈操作持续时间，显示计算机正在工作，没有停滞或者报错。对于用户来说，这种回应与提示是对用户操作的认可与尊重，更有助于用户确定下一步操作该如何进行。

（7）错误提示：关键操作完成后或数据删除等操作前给出明确提示，操作错误或系统出现错误时，给出的出错信息中提供差错产生的原因，并指示如何进入正确的步骤，帮助用户从错误中恢复。

### 7.2.3 用户界面测试

由于用户界面是用户与软件打交道的唯一渠道，用户界面是否友好在很大程度上决定了软件的易用性，因此用户界面测试是软件易用性测试最重要的一项内容。下面从界面整体测试、界面元素测试和输入测试三方面讨论用户界面的测试。需要指出的是，由于界面是否友好与测试人员的主观感受有关，界面测试通常需要多个测试人员对同样的界面进行测试，以减少个人主观感受对测试结果的影响。

**1．界面整体测试**

界面整体测试是指测试人员对界面的整体观感进行体验，从整体上检查界面的友好性。通常，界面整体测试包括合理性测试、一致性测试和规范性测试三个方面。

（1）合理性测试。

所谓合理性是指界面是否与对应的功能融洽。合理性首先是功能方面的，即界面应该辅助功能的展现。例如，信息的显示需要与背景明显区分，以免被用户忽略。其次，合理性还表现在情绪方面。界面的色彩、布局等会对用户的情绪造成影响，设计合理的界面能够保证用户心情舒畅地使用软件，减少用户出错的概率。对于功能方面的合理性，测试时可以查看各个界面进行评估；对于情绪方面的合理性，测试时通常需要多个用户参与评估，并对各个用户的测试结果进行汇总。

（2）一致性测试。

用户在使用软件时通常会依据已有的使用经验来推测未使用过的功能的使用方式，如果不同功能在使用上风格不一致，用户发生误用的概率就会增加，而用户为了减少误用就必须花费更多的精力学习和记忆。所谓界面一致性测试，其主要目的就是检查软件在完成不同功能时界面的风格是否一致。通常，测试人员需要检查这几个方面的一致性：窗口风格的一致性；窗口布局的一致性；提示信息的风格和措辞的一致性；操作方式的一致性；颜色使用的一致性；快捷键使用的一致性。

需要指出的是，界面风格有时会因环境差异而出现差异。例如，同一个软件在不同分辨率下显示效果可能会差异很大。对需要在不同环境下运行的软件而言，界面的一致性测试应该还包括在不同环境下界面风格的兼容性测试。

（3）规范性测试。

规范性可以看做是一致性的扩展。一致性通常指同一软件内界面风格的一致性，而界面风格的规范性是指同类软件间界面风格的一致性。例如，一套窗口系统通常有一套界面风格（如 Microsoft Windows 风格和 Apple Macintosh 风格）。另外，针对不同行业或者不同用户群也可能存在行业界面风格或者用户群界面风格。规范性测试的方法通常与一致性测试的方法类似，只是此时的测试人员需要预先了解同类软件的界面风格。

**2．界面元素测试**

测试用户界面时除了测试界面的整体风格外，还需要对界面上的各种元素的易用性进行测试。常见的测试内容包括窗口测试、菜单测试、文字测试和图标测试。

（1）窗口测试。

窗口是大多数软件最基本的界面元素，窗口测试也是界面元素测试的最基本内容，其内容主要包括两个方面：

- 对窗口本身操作的测试，如窗口的移动、缩放、最大化、最小化等。由于大多数软件基于特定的窗口系统，这方面通常没有问题。
- 对窗口中内容操作的测试，如窗口中各种控件的布局、窗口中文字信息的显示等。这是窗口测试的重点。

（2）菜单测试。

菜单的目的是为了组织软件的功能。由于大部分软件的菜单主要基于相应窗口系统的菜单机制，菜单测试的重点不是菜单风格，而是菜单的内容。好的菜单能够帮助用户使用软件，而坏的菜单反而会干扰用户的正常使用。菜单测试时，测试人员通常需要检查两个方面：

- 菜单的组织，即菜单是否按照用户的使用习惯和菜单项间的逻辑联系进行组织。
- 菜单项的措辞，即菜单项上的文字是否能准确反映菜单对应的功能。

（3）文字测试。

文字测试就是检查界面上显示的文字是否能够帮助用户使用软件。文字测试通常包括两个方面：

- 文字的内容，即文字能否准确表达软件想要传达的信息，以及文字中是否存在语法和拼写错误等。
- 文字的显示风格，即文字的颜色、大小、位置等是否能够让用户方便地理解文字所传达的信息。

（4）图标测试。

大多数软件使用图标作为常用功能的快捷方式。图标测试时，主要检查图标是否能够准确反映所对应的功能，从而方便用户的使用。由于图标测试比较主观，测试时通常需要综合多个测试人员的评估意见。

**3．输入测试**

对用户而言，能否方便地使用输入设备进行输入也是软件易用性的一个重要方面。首先，不同的输入设备具有不同的特点，好的软件能够综合运用多种设备完成输入。由于许多应用软件不直接管理输入设备，这方面的易用性在很大程度上取决于应用软件所基于的窗口系统，通常问题不多。但对于直接管理输入设备的应用软件，则需要详细测试软件对于输入设备的使用是否方便。其次，软件在不同环境中运行时，输入设备可能会有差异，在不同的环境中用户都应该能够方便地进行测试。例如，有的环境中没有鼠

标，而有的环境有；不同环境中鼠标的类型也会不同，有单键鼠标、双键鼠标、三键鼠标，有的鼠标有滑轮，有的则没有。测试时，需要针对各种可能的环境进行检查，确认软件能够很好地支持各种输入设备。

### 7.2.4 兼容性测试

在实际的软件开发中，软件通常都需要在多种不同的软硬件环境中运行。然而，由于任何一个软件都或多或少地依赖其所运行的环境，环境的差异可能导致软件在不同的环境下运行会有不同的表现。因此，需要针对软件对其运行环境的依赖进行测试，以验证软件是否能够在所有期望的环境中运行，这就是所谓的兼容性测试。需要指出的是，兼容性的具体含义与期望的运行环境密切相关。如果期望的运行环境存在众多的可变性，则兼容性的含义就很丰富，兼容性测试也很复杂；反之，兼容性就很可能不存在问题，兼容性测试也会很简单。例如，对于文字处理软件的兼容性，除了要考虑与硬件和操作系统的兼容性外，通常会包括与其他文字处理软件的互操作性；而对于其他的软件则可能根本没有互操作的问题。在通常情况下，兼容性测试主要需要考虑硬件、软件和数据3个方面。

**1．硬件兼容性测试**

由于大多数软件并不直接与硬件交互，而是通过操作系统与硬件交互。因此，如果能保证操作系统与硬件的兼容性以及软件与操作系统的兼容性，通常可以保证软件与硬件的兼容性。也就是说，硬件兼容性测试通常并不需要盲目地把软件在各种不同的硬件环境下运行和测试，而只需要考虑以下两个方面。

- 不同的硬件配置可能影响软件的性能，因此需要有针对性地进行测试。
- 如果软件使用了某些硬件的特定功能，一般需要对此进行兼容性测试。

硬件兼容性测试的具体内容如下。

（1）与整机的兼容性。

与整机的兼容性测试主要考虑软件的性能以及软件在压力下的运行情况。因此，需要针对常见的硬件配置进行测试，以确定软件能够在多种硬件配置下运行。另外，如果软件对硬件配置的要求比较高，通常还要测试软件对硬件配置变化的敏感度。例如，当硬件配置略有下降时，软件的性能和稳定性是否急剧下降。

（2）与板卡及外部设备的兼容性。

如果被测软件需要直接访问某类板卡或外部设备（如摄像头），通常都需要针对这些板卡或设备进行兼容性测试。此时，通常需要针对软件中所有访问这些板卡或设备的接口调用进行测试，以确保对这些接口的访问适用于所有型号的板卡和设备。特别地，如果软件中使用了非标准的接口，测试时除了需要访问该接口外，通常需要进一步对该接口的多种输入组合进行测试。需要注意的是，某些板卡和外部设备的兼容性问题可能不在功能上而是在性能上，测试时需要为此产生一定的压力。侧如，显卡的显存大小会

影响显卡的刷新速度，测试时需要使用一些对刷新速度要求高的测试用例。

**2．软件兼容性测试**

软件在运行过程中通常需要或多或少地与其他软件进行交互，任何交互问题都可能引起该软件的运行问题。因此，通常需要针对与该软件可能发生交互的其他软件的兼容性进行测试。一般来说，软件兼容性测试通常要考虑以下方面。

（1）与操作系统的兼容性。

如果一个软件承诺可以在多种操作系统上运行，就需要测试该软件与操作系统的兼容性。一般来说，大多数软件不能在完全不同的操作系统上运行，因此也无须测试这方面的兼容性。

兼容性问题往往出现在同一操作系统平台的不同版本上。例如，通常说的 Windows 平台一般包括 Windows 98、Windows ME、Windows 2000、Windows 2003、Windows XP、Windows Vista 等，在测试时需要针对以上每种 Windows 系统测试软件的兼容性。由于每种 Windows 系统又可以分为多个版本，完全相同的两个操作系统打了不同的系统补丁又可以被认为是具有微小差别的不同版本，如果认为这些差别有可能影响软件的运行，甚至还要对这些版本进行兼容性测试。

对于其他的操作系统，情况可能会好一些，但兼容性测试也不可避免。UNIX 和 Linux 是两种类似的操作系统，如果一个软件需要在这两种操作系统上运行，通常首先需要为此进行兼容性测试。由于 UNIX 和 Linux 又各自有多个版本（包括不同厂商的发行版本以及不同时期的版本），又需要针对这些版本进行测试。

（2）与数据库的兼容性。

目前常用的数据库产品大多是支持 SQL 标准的数据库，如 MS SQL Server、Sybase SQL Server、Oracle、ODBC、JDBC 等。但不同数据库对 SQL 标准（实际上，SQL 标准又有多个版本，最常用的 SQL—92 标准又包含 4 个一致性等级）的支持又各有不同。因此，如果软件需要支持不同的数据库，通常需要针对不同的数据库产品进行兼容性测试。另外，同一个数据库产品实际上又包含多个版本，如 Oracle 8i 和 Oracle 9i，此时往往也需要对这些版本进行兼容性测试。需要指出的是，如果软件支持 ODBC 或 JDBC，通过 ODBC 或 JDBC 与实际的数据库连接，由于 OBDC 和 JDBC 只能部分屏蔽实际数据库的异构性，此时的兼容性测试应该既包括对 ODBC 或 JDBC 的测试，又包括对实际数据库的测试。

（3）与浏览器的兼容性。

对于 Web 应用来说，软件必须通过浏览器来使用，对于不同的浏览器以及浏览器的不同版本很可能出现兼容性问题。通常出现的兼容性问题包括：

- 某些特定的 HTML 标签只能在某些浏览器版本上使用，特别是 HTML 的版本也在变化，符合最新 HTML 版本的页面很可能会在某些浏览器上不能正确显示。
- 某些脚本和插件可能只适用于特定的浏览器，例如 ActiveX 是微软的产品，现有

的浏览器中只有 Internet Explorer 支持。

- 不同浏览器对于安全性的设置各有不同，需要测试不同浏览器是否都能够为使用该 Web 应用提供合适的安全设置。

（4）与中间件的兼容性。

中间件是近年来逐渐兴起的一类新的系统软件，目前越来越多的应用软件需要中间件的支持才能运行。一般而言，不同厂商开发的中间件产品差别比较大，在一种中间件上运行的软件很难在其他中间件上运行。因此，与中间件的兼容性测试主要针对的是同一产品的不同版本。另外，对基于 J2EE 的中间件而言，由于都需要符合 J2EE 规范，不同中间件具有一定兼容性。因此，可能某些应用软件需要在多种不同的 J2EE 中间件上运行，例如同时在 WebLogic 和 WebSphere 上运行。此时的兼容性测试应该主要集中在各个 J2EE 中间件特有的服务上。

（5）与其他软件的兼容性。

测试应用软件时，除了要考虑它与操作系统、数据库、中间件等系统软件的兼容性外，还需要考虑它与其他软件的兼容性。通常需要考虑以下三种情况：

- 与其他支撑软件的兼容性。实际上，支撑软件的作用类似于系统软件，只是支撑软件通常只是某个领域应用软件的运行基础。此时的兼容性测试与上面讨论的与系统软件的兼容性测试类似。
- 与其他同类软件的兼容性。此时主要测试两个方面。第一个方面是同类软件间的冲突：同类软件通常会使用相同的系统资源，当多个同类软件并存时，可能会相互排斥。例如，一个杀毒软件可能会影响另一个杀毒软件的正常工作。第二个方面是同类软件间的互操作：当存在多个同类软件时，用户一般希望它们之间可以互操作，用户因此可以充分利用各个软件的优势。
- 与不同类软件的兼容性。不同类的软件间通常不存在交互和冲突，因此通常不存在兼容性问题，但对于一些特殊情况需要加以考虑。例如，杀毒软件在查杀病毒时可能会破坏某些文件的格式信息，造成相应软件不能正常运行。此时的测试需要充分考虑不同类软件间存在的特殊交互或冲突关系。

（6）平台软件的兼容性测试。

上面讨论主要集中在应用软件的兼容性测试上。实际上，平台软件的兼容性测试通常比应用软件的兼容性测试更为复杂和困难。平台软件和普通的应用软件一样，也需要在合适的软硬件环境中运行，因此平台软件兼容性测试的内容通常需要包含应用软件兼容性测试的基本内容，如硬件兼容性测试、软件操作系统兼容性测试、数据库兼容性测试等。由于平台需要支持其他软件的开发和运行，平台软件的兼容性还需要包括与其所支持的软件的兼容性。

一般而言，可以把平台软件分为运行平台和开发平台两类。对于运行平台，兼容性测试还需要包括测试平台软件与其上运行的应用软件的兼容性。此时的测试与前面介绍

的操作系统、数据库以及中间件软件的兼容性测试类似，只是此时的兼容性测试是多个应用软件与一个平台软件的兼容性，而前面介绍的兼容性测试是一个应用软件与多个平台软件的兼容性。由于需要考虑的应用软件可能会很多，此时的测试将是个很烦琐的过程。

对于开发平台，兼容性测试还包括测试所开发的软件与相应环境的兼容性。例如，测试一个中间件应用的开发平台时，如果该开发平台需要兼容 WebLogic 和 WebSphere，则需要测试该平台开发出来的软件（特别是平台生成的代码）与这两个中间件的兼容性。

**3．数据兼容性测试**

对用户而言，数据出现问题往往比软件本身出现问题更严重。例如，一位作家使用某种文字处理软件辅助进行文学创作，如果他某天发现记录其半年创作内容的文件无法打开，这可能比其在使用该文字处理软件时遇到的其他软件错误都要严重得多。因此，数据兼容性通常也是兼容性测试关注的一个非常重要的方面。常见的数据兼容性主要包括以下两类：

（1）不同版本间的数据兼容性。

一个软件系统通常会在其生命周期中出现一系列版本，例如 Windows 操作系统包括 Windows 98、Windows 2000、Windows 2003、Windows XP、Windows Vista 等。一般而言，新版本出现时，旧版本已经使用了比较长的一段时间，在其使用期间一般已经积累了大量历史数据，有些历史数据甚至是新版本顺利启动的必要数据。因此，测试新版本软件是否能够兼容旧版本的数据是数据兼容性测试的一个重要方面。如果新版本与旧版本使用相同的数据格式，此时的测试主要针对新版本是否能兼容此数据格式。如果新版本需要将旧版本的数据转移到新的数据格式，还需要测试在数据转移中是否会出现冗余甚至是垃圾数据。另外，有时新版本需要同时支持新旧两种数据格式，也必须对此进行测试。

（2）不同软件间的数据兼容性。

数据兼容性测试不但存在于同一软件的不同版本间，也存在于不同的软件之间。一种常见的情况是一个系列中的不同软件通过约定好的数据格式实现集成；另一种情况是不同的软件通过标准的数据格式（如 XML）进行集成。此时的测试要针对相应的一种或多种数据格式，检查被测软件是否能够通过复合数据格式的各种数据正确进行交互。

### 7.2.5　安装测试

除了嵌入式软件之外，安装是软件产品实现其功能的第一步。对于一般的应用软件来说，最早体现其易用性的就是软件安装。现在的软件系统越来越庞大，有可能使安装过程变得复杂，安装耗时也会越来越长。没有正确的安装根本就谈不上正确的使用，因此安装测试就显得尤为重要，安装的易用性是安装测试的主要内容。

安装测试的方法很简单，就是按照用户安装手册安装软件，来评估安装过程的易用

性、正确性。那么对于安装测试需要注意一些什么呢，我们认为至少应该从以下几个方面来考虑。

- 安装手册的评估。在安装前需要检查安装手册或用户文档中的安装说明，一般来说，安装手册需要对安装平台、安装过程需注意的事项以及需手动配置的部分进行详细说明。
- 安装的自动化程度测试。由于制作安装程序的软件很多，其中很成熟的有 Installshield 等，很多软件采用了自动安装的方式。但由于部分软件的特殊性，有时必须采用一定的手动配置来完成安装。我们要评估软件安装过程的自动化程度。一般来说，软件的安装程序尽量要做到“全自动化”，即使在不得已的情况下需要进行手动配置，也要采取一些措施，比如选择框方式等，使手动配置变得简便和明确。
- 安装选项和设置的测试。在安装过程中常常需要对安装的项目进行选择，也可能要设置不同的信息，比如安装路径等。安装测试时需要对不同的选项和设置方案进行测试，验证各种方案是否都能安装成功。
- 安装过程的中断测试。一个大型的软件有可能需要数小时来进行安装，如果因为断电、文件冲突或读写错误导致安装过程的非正常中断，有可能使已进行的安装工作前功尽弃。一个好的自动化安装程序应该能记忆安装的过程，当恢复安装时，安装程序能自动进行检测，并从“断点”继续安装。
- 安装顺序测试。对于大多数应用系统，特别是分布式系统，常常需要安装软件系统的不同组成部分。不同的安装顺序常常会导致安装失败，或者会引起一些不可预料的错误，例如，先安装客户端后安装服务器，会导致某些软件的客户端与服务器连接不上。如果《安装手册》中未明确指出安装顺序，则需要测试不同顺序的安装过程。
- 多环境安装测试。不同的应用环境下安装的情况也是不一样的，我们至少要在标准配置、最低配置和笔记本电脑三种环境中进行安装测试。很多情况下产品声称的最低配置并不符实，所以最低配置环境测试是非常必要的。另外，有些系统级的软件常常在笔记本电脑上安装时发生错误，例如，由于笔记本电脑的高集成度特性，Linux 桌面操作系统在笔记本安装时出现硬件兼容性问题。
- 安装的正确性测试。在上述的安装测试后，都需要进行简单的使用以验证安装的正确性。另外，还要考察对其他应用程序的影响。
- 修复安装测试与卸载测试。修复安装测试指软件使用后，根据需要添加或删除软件的一些组件或者修复受损的软件。修复安装和卸载也应该是自动化的，通常情况下，安装、修复安装以及卸载是一个完整安装程序中的不同选项。进行修复安装测试时，需检查修复对软件有无不良的影响，例如，修复可能造成系统数据丢失。卸载测试重点检查卸载是否完全，不能完全卸载时有无明确提示信息等。

## 7.2.6　文档测试

### 1．文档测试的范围

软件产品由可运行的程序、数据和文档组成。文档是软件的一个重要组成部分。

在软件的整个生命周期中，会用到许多文档，在各个阶段中以文档作为前阶段工作成果的体现和后阶段工作的依据。在软件的开发过程中，软件开发人员需根据工作计划和需求说明书由粗而细地进行设计，这些需求说明书和设计说明书构成了开发文档。为了使用户了解软件的使用、操作和对软件进行维护，软件开发人员需要为用户提供详细的资料，这些资料称为用户文档。而为了使管理人员及整个软件开发项目组了解软件开发项目安排、进度、资源使用和成果等，还需要制订和编写一些工作计划或工作报告，这些计划和报告构成了管理文档。软件文档的分类结构图如图 7-1 所示。

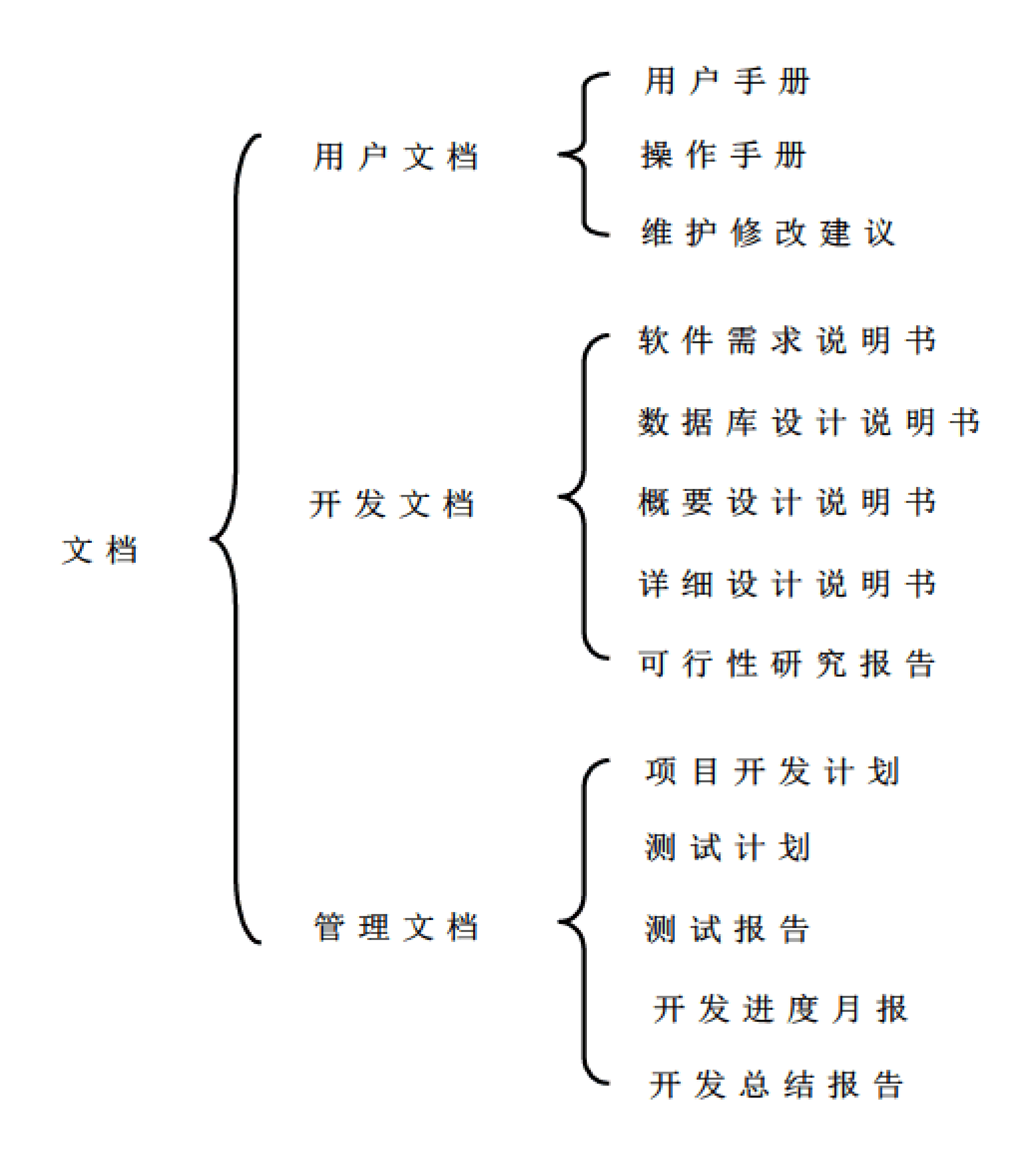



图 7-1　软件的文档分类结构图

下面对这些文档进行一些说明。

- 可行性研究报告：说明该软件开发项目的实现在技术上、经济上和社会因素上的可行性，评述为了合理地达到开发目标可供选择的各种可能实施的方案，说明并论证所选定实施方案的理由。
- 项目开发计划：为软件项目实施方案制订出具体计划，应该包括各部分工作的负责人员、开发的进度、开发经费的预算、所需的硬件及软件资源等。项目开发计划应提供给管理部门，并作为开发阶段评审的参考。
- 软件需求说明书：也称软件规格说明书，其中对所开发软件的功能、性能、用户

界面及运行环境等作出详细的说明。它是在用户与开发人员双方对软件需求取得共同理解的基础上达成的协议，也是实施开发工作的基础。

- 数据库设计说明书：只对使用数据库的软件适用，该说明书应给出数据库的整体架构及各个数据表中的逻辑关系。
- 概要设计说明书：该说明书是概要设计阶段的工作成果，它应说明功能分配、模块划分、程序的总体结构、输入输出以及接口设计、运行设计、数据结构设计和出错处理设计等，为详细设计奠定基础。
- 详细设计说明书：着重描述每一模块是怎样实现的，包括实现算法、逻辑流程等。
- 用户手册：本手册详细描述软件的功能、性能和用户界面，使用户了解如何使用该软件。
- 测试计划：计划应包括测试的内容、进度、条件、人员、测试用例的选取原则、测试结果允许的偏差范围等。
- 测试分析报告：测试工作完成以后，应提交测试计划执行情况的说明。对测试结果加以分析，并提出测试的结论意见。
- 开发进度月报：该月报是软件人员按月向管理部门提交的项目进展情况报告。报告应包括进度计划与实际执行情况的比较、阶段成果、遇到的问题和解决的办法以及下个月的打算等。
- 项目开发总结报告：软件项目开发完成以后，应与项目实施计划对照，总结实际执行的情况，如进度、成果、资源利用、成本和投入的人力。此外还需对开发工作作出评价，总结出经验和教训。
- 操作手册：本手册为操作人员提供该软件各种运行情况的有关知识，特别是操作方法的具体细节。
- 维护修改建议：软件产品投入运行以后，发现了需对其进行修正、更改等的问题，应将存在的问题、修改的考虑以及修改的估计影响作详细的描述，写成维护修改建议，提交审批。

以上这些文档是在软件生存期中，随着各阶段工作的开展适时编制的。其中有的仅反映一个阶段的工作，有的则需跨越多个阶段。

**2．用户文档的内容**

当用户文档仅包含一个 Readme 文件时，文档的测试只需要对其进行拼写检查，确认其中涉及的技术准确无误，最多对 Readme 文件进行病毒扫描，确保其不带病毒就足够了。

但随着技术的进步和市场的规范，用户文档的范围越来越大了。以下这些都可以算是用户文档。并不是每一个软件都必须具有所有这些文档，但大多不出此列。

- 包装上的文字和图案。包括纸盒、包装纸或信封等。这些包装上可能含有软件的屏幕截图、特性清单、系统要求和版权信息等。

- 宣传材料、广告及其他插页。这些是软件开发者促进其相关软件销售的重要工作，同时提供补充内容、其他软件介绍和服务联系方式等。虽然对于一些用户来说，可能会随手丢弃，但同样也有一些用户会严肃地对待它们，因此这些信息也必须正确。
- 授权/注册登记表。这是希望用户填写内容、注册软件并寄回的卡片，它同样作为软件的一部分，也可能是电子文档，让用户在线阅读及注册。
- 最终用户许可协议。这是要求用户在使用软件前认可的一份法律文书，包括用户同意不得复制软件等内容。最终用户许可协议可能是打印在软盘或光盘的盒子上，或信封上，也有可能是在安装过程中弹出一个窗口显示在屏幕上。
- 标签和不干胶条。这类文档可能是软盘或光盘上的标签，或出现在包装盒上、印刷的材料上，像序列号、信封或光盘盒的封口标签等。这上面的内容也是需要测试的。
- 安装和设置指导。对于简单的软件来说，可能是包装中的一页纸，对于复杂软件来说，就有可能是完整的一本手册了。
- 用户手册。由于电子手册的实用性和灵活性，使纸介质的手册已大大减少了。目前一些软件附带简明的“入门”类小册子，而详细信息往往采用电子文档。电子手册大都以光盘形式随软件销售，也有的在网站上供用户下载。
- 联机帮助。联机帮助与用户手册有时可以互换使用，甚至取代用户手册。由于联机帮助可以有索引和搜索功能，一些联机帮助允许模糊查询或多关键字查询，更方便了用户查找所需信息。
- 指南、向导。这些工具已不仅仅是一页一页的文档，它们是文本内容和程序的结合，通常属于联机帮助的一部分。用户可以提出问题，然后向导将一步步引导用户完成任务。关于向导的例子可以参考微软 Office 的助手功能。
- 样例、示例和模板。字处理、网页制作等软件往往带有表单或样例，用户只需填写相应内容即可快速达到具有专业外观的效果。编译器可以用几段代码来演示如何使用编程语言的某些方面。财务软件可以通过模拟账套来解释软件的使用方法。
- 错误提示信息。这虽然是软件程序的一部分，但在程序测试中往往被忽略，而它们也是文档的一部分。

**3. 用户文档的作用**

对于软件测试人员来说，对待用户文档要像对待程序一样给予同等关注和投入，因为对于用户来说，文档和程序同样重要。

充分有效的文档有如下优点。

- 改善易安装性。用户需要将软件产品安装到自己的计算机上。他们可能需要做复制文件、配置数据库、输入初始参数、将以前的数据导入等大量工作。安装程序

是最后编写的，开发人员可能不会像对待产品的其他部分一样认真对待它，因为一些人认为用户仅会安装程序一两次而已，因此安装程序得到的测试和开发支持最少。然而，用户对产品的初次体验是从软件的安装开始的，如果在安装时遇到了困难，用户可能会对软件失去信心，或提出昂贵的技术支持要求，或干脆放弃使用软件。清晰、正确的安装指南是产品文档中最为重要的部分之一。

- 提高软件的易用性。具备优良文档的产品更易于使用。文档编制得越好，用户对产品的理解就越快，操作中发生的理解方面的错误就越少，效率就越高。高效的文档往往是面向任务的，它会估计用户意图，并说明如何完成各项任务。并不是说面向特征的用户手册就不好，面向特征的用户手册会独立地描述功能特征，按菜单顺序，甚至按字母顺序逐个描述功能。不同的软件类型适用于不同的手册类型，一些软件会同时提供这两类手册。
- 改善软件可靠性。不清晰、不正确的文档会降低产品的可靠性，用户使用它容易出现操作上的错误。优秀的文档即使在程序设计得很糟糕的情况下，也能有助于减少用户犯错次数。
- 促进销路。高质量的文档常会被作为卖点，可以帮助销售人员说明和推荐产品。在很多软件评审中它也扮演着重要的角色。
- 降低技术支持的费用。由用户发现问题比在产品开发早期发现问题的修复费用要高出数十倍。好的文档能够通过恰当的解释引导用户自己解决问题，尽可能地避免用户打技术支持电话。如果用户文档描述了程序不具备的功能，开发商就是在作虚假的宣传；如果用户文档描述了实际上无法执行的操作，则开发商是在误导用户。不正确的指导会浪费用户不必要的时间和精力，也给开发商增加了法律上的风险。

**4．用户文档测试需要注意的问题**

对于软件用户来说，程序之外的部分也是软件的一部分，他们并不管这些东西是由程序员、作家还是图形艺术家创建的。他们关心的是整个软件包的质量。

文档测试中需要注意如下问题。

- 文档常常得不到足够的重视，文档的开发缺乏足够的资金和技术支持，而文档的测试更得不到重视。一个好的软件项目，一定要为文档测试留出预算，像对程序一样对文档给予关注。对文档中发现的缺陷，也需像发现程序缺陷一样给出报告。
- 编写文档的人可能并不是软件特性方面的专家，对软件功能可能了解得并不深入。其结果就是写出来的产品说明书可能并不到位，或者不能解释复杂的产品特性。软件文档测试人员可以与文档作者紧密合作，保证文档中所包含信息的质量，并随着产品的更新而更新。更重要的是，测试人员可以发现并指出程序中难以使用或难以理解之处，让文档作者在文档中作出更好的解释。
- 由于文档的印刷需要花费不少的时间，可能是几周，如果追求印刷质量的话可能

需要几个月。而在这段时间发现的错误可以有时间修改，程序很可能已经发生了改变，而文档无法反映最终的修改。Readme 文件的发明正因为如此，它是将最后的改动通知用户的方式。它能使文档保持到最后一刻发布，从而保证与软件程序的同步。同时，随软件发布的联机帮助信息也可以尽可能地反映最新的修改。如果文档测试不够充分，大量的错误将不得不随着印刷精美的手册到达用户手中，而 Readme 文件就不是最新特性的发布而是长长的勘误表了。

- 文档测试不仅仅是对文字的校对，更可以辅助找到更多的程序错误。文档编写人员与文档测试人员审视程序的角度与程序员和程序测试人员并不相同，因此由文档测试揭示的问题也不同于程序员和程序测试人员所发现的问题，文档测试往往会发现其他测试无法发现的严重错误，例如，功能实现错误、易用性不好、用户手册与程序实现不吻合等问题。当然，这是在全面测试的基础上，而全面测试意味着每 3～5 页花费 1 小时的时间。测试人员审看文档的速度越快，从文档和程序中发现问题的机会就越少。加强测试监督、重新培训测试人员，甚至更换测试人员能有助于解决这一问题。

**5. 用户文档测试的要点**

文档测试分为两类，如果是非程序，例如打印的手册或产品包装盒，其测试可以视为技术校对。如果文档和程序紧密结合，例如超链接形式的电子手册或联机帮助，或助手一类的帮助系统，就要进行与程序测试类似的测试。

无论是文档或是程序，作为文档测试人员，都必须像用户那样对待它，应该说像最仔细的用户那样，认真阅读，跟随每个步骤，检查每个图形，尝试每个示例。只有这样，才能尽可能找出软件和文档中的缺陷。

文档测试中，对于如下几个方面需要特别关注。

- 读者群。文档面向的读者定位要明确。对于初级用户，可能需要从鼠标的用法、点击确定按钮等讲起；对于中级用户，重要界面的截图和关键步骤每一个参数的选择方法都需要介绍；对于高级用户，则没有必要给出太多的界面截图，但对重要参数的讲解一定要深入，用词要专业。特别是不论用户群定位如何，文档都不可以写成散文、诗歌或者侦探、言情小说，文档的目的是要让用户看得懂，能理解。
- 术语。文档中用到的术语要适用于定位的读者群，用法一致，标准定义与业界规范相吻合。如果有索引或交叉引用，所有的术语都应能够进行索引和交叉引用。如果术语较多，在纸介质手册的末尾应给出术语索引；如果被测软件提供二次开发功能，有大量函数，则有必要编写独立的函数手册和开发指南。
- 正确性。这是非常重要的，也会占用文档测试的大量时间和人力。测试中需检查所有信息是否真实正确，查找由于过期产品说明书和销售人员夸大事实而导致的错误。检查所有的目录、索引和章节引用是否已更新，尝试链接是否准确，产品

支持电话、地址和邮政编码是否正确。

- 完整性。慢慢地仔细阅读文字，完全根据提示进行操作，不要作任何假设。对照软件界面检查是否有重要的分支没有描述到，甚至是否有整个大模块没有描述到，耐心补充遗漏的步骤。用户不会知道遗漏了什么信息，直到有一天他使用软件时走到了这个分支。对于极其熟悉被测软件的人来说，这项测试相当困难，因为思路已固定地按照一定的流程去测试，极易忽略不常用的部分。因此，可以考虑让不是很熟悉被测软件的人员进行此项目的测试。
- 一致性。按照文档描述的操作执行后，检查软件返回的结果是否与文档描述相同。要留意软件界面上出现的版本号与手册、帮助上的信息是否一致。
- 易用性。纸介质文档可以通过目录、关键词索引提高用户使用的易用性。条理清晰、结构合理的文档是优质软件的一个显著特征。对关键步骤以粗体或背景色给用户以提示，合理的页面布局、适量的图表都可以给用户更高的易用性。电子文档或帮助系统显然比纸介质在这方面有更大的优势。需要注意的是，文档要有助于用户排除错误，只描述正确操作而不描述错误处理办法的文档是不负责任的。与程序大多用于错误处理一样，文档对于用户看到的错误信息应当有更详细的文档解释，而且不应让用户花费太多的时间去寻找所需的解释。
- 图表与界面截图。检查所有图表与界面截图是否与发行版本相同。对于成熟的软件开发商来说，界面在设计阶段就应基本确定，不应在软件开发后期有大的变动。而此项测试就是要发现在文档完成后是否有界面变动，确保屏幕截图源于发行版本。测试中还要注意图表标题的正确性。
- 样例和示例。像用户一样载入和使用样例。如果是一段程序，就输入数据并执行它。以每一个模板制作文件，确认它们的正确性。想象一下样例不能执行的问题交给技术支持人员时的情景……
- 语言。对于英语文档来说，拼写和语法检查器太常见了，一般不会出现拼写和语法错误。但对于中文文档来说，可以采用一些校对工具辅助人工检查，并进行细致专业的校对，不要让用户发现错别字，不要出现有二义性的说法。特别要注意的是屏幕截图或绘制图形中的文字，不能想当然，没有任何工具能够从图形中找出语法错误。
- 印刷与包装。文档测试似乎完成了，文档终于变成精美的册子，这时，测试人员还需抽查印刷质量，看看手册厚度与开本是否合适，翻看起来是否方便，包装盒的大小是否合适，光盘盒的固定有没有问题，有没有零碎易丢失的小部件等等。这时发现的问题，如果不是太严重，已不可能在这个版本中进行改进，但对下一个版本的制作来说是非常有价值的。

**6. 针对用户手册的测试**

用户手册是用户文档中最重要的一部分。在对用户手册进行测试时，应拿着它

坐在计算机前，进行如下操作。

- 准确地按照手册的描述使用程序。在每个例子中如实地进行每个键盘操作。用户在按照手册运行程序时可能会进行错误的操作，因此测试时测试人员也可以随心所欲地“犯错误”。检查计算机对错误的处理和手册对错误处理的描述应当占用测试人员的大部分精力。
- 尝试每一条建议。即使建议并没有完全表达清楚，仍应按步骤去尝试。用户依照建议会做什么，测试人员就应当做什么，甚至尝试更多的可能性。
- 检查每条陈述。测试人员需要对每条陈述进行检查，因为用户手册是产品最终的规范，是用户检查程序运行是否正确首先求证的地方。
- 查找容易误导用户的内容。有些示例和特征描述得并不准确，一般的读者可能会从中归纳出错误的结论。用户可能对程序的能力抱有过高的期望，或是凭空想象一些实际并不存在的约束条件。尽早标识出易被人误解的内容，这一点极其重要。

**7. 针对在线帮助的测试**

帮助文档的测试在很大程度上与用户手册测试相同，但帮助并不只是用户手册的电子版，因此再给出以下几点补充说明。

- 准确性。对帮助准确性检查的细致程度至少要接近于对用户手册的检查。通常帮助文本都没有得到良好的处理和充分测试，无法受到用户的欣赏。一旦用户发现帮助中存在明显的错误，他可能对帮助系统的信任程度大大下降。
- 帮助是文档编写和程序编写的结合。不仅要检查文本的准确性，还要检查程序的可靠性。制作帮助的人员往往并不是专业的程序员，他们在使用帮助制作工具的技巧、与程序的接口等问题上不可能达到完美。
- 帮助索引。如果帮助系统包含了索引或主题列表，允许用户由索引进入到主题中，测试人员就必须逐条进行检查。
- 超链接。除非是早期开发的软件或开发商对帮助过于不重视，超链接是在线帮助中必需的功能。测试人员必须对每个链接都测试到，复杂的超链接可能会对一个主题形成树状结构展开的若干页面，甚至构成网状结构。测试人员有义务检查每条分支。
- 链接的意义。索引和链接的条目应当是有意义的，测试人员需要发现是否有一些帮助主题未出现在索引里，或出现的名称不恰当。如果用户不能迅速找到所需的信息，只能说帮助系统在一定程度上是失败的。
- 帮助的风格。很少会有用户能悠闲地查看帮助，帮助的阅读者是带着问题、焦躁不安而缺乏耐心的。帮助文本需要比用户手册更为简洁，风格也应更为简单。良好的帮助系统应该是面向任务或面向操作的，它必须提供一些有意义的信息，让用户能立即开始或继续他的操作。任何在帮助中出现的令人迷惑或离题的内容都可以作为测试问题。

# 第 8 章　验收测试阶段

**学习目标：**

1．了解验收测试的概念
2．熟悉验收测试常用的策略
3．了解验收测试的总体思路

## 8.1　引言

软件经过系统测试之后，在实验室环境中的测试任务已经完成，然而在交付用户使用之前，还有最后一步需要完成，那就是验收测试。为了保证软件的功能、性能等指标能符合最终用户的需求，软件还需要经过验收测试这一步骤，下面我们来具体介绍这一环节。

## 8.2　验收测试

### 8.2.1　验收测试的概念

在通过了系统的功能测试及软件配置审查之后，就应该开始系统的验收测试。验收测试是部署软件之前的最后一个测试操作。验收测试的目的是确保软件准备就绪，并且可以让最终用户将其用于执行软件的既定功能和任务，是向未来的用户表明系统能够像预定要求那样工作。

验收测试是以用户为主的测试。软件开发人员和质量保证人员也应参加。由用户参加设计测试用例，使用用户界面输入测试数据，并分析测试的输出结果。一般使用用户工作生产中的实际数据进行测试。在测试过程中，除了考虑软件的功能和性能外，还应对软件的可移植性、兼容性、可维护性、错误的恢复功能等进行确认。将程序的实际操作与原始合同进行对照。

### 8.2.2　验收测试标准

验收测试应检查软件能否按合同要求进行工作，即是否满足软件需求说明书中的确认标准。

实现软件确认要通过一系列黑盒测试。验收测试同样需要制订测试计划和过程，测

试计划应规定测试的种类和测试进度，测试过程则定义一些特殊的测试用例，旨在说明软件与需求是否一致。无论是计划还是过程，都应该着重考虑软件是否满足合同规定的所有功能和性能，文档资料是否完整、用户界面和其他方面（如可移植性、兼容性、错误恢复能力和可维护性等）是否令用户满意。验收测试的结果有两种可能，一种是功能和性能指标满足软件需求说明的要求，用户可以接受；另一种是软件不满足软件需求说明的要求，用户无法接受。项目进行到这个阶段才发现严重错误和偏差一般很难在预定的工期内改正，因此必须与用户协商，寻求一个妥善解决问题的方法。

### 8.2.3　验收测试过程

**1．软件需求分析**

了解软件功能和性能需求、软硬件环境要求等，并特别要了解软件的质量要求和验收要求。

**2．配置复审**

验收测试的另一个重要环节是配置复审。复审的目的在于保证软件配置齐全、分类有序，并且包括软件维护所必需的细节。

**3．编制《验收测试计划》和《项目验收准则》**

根据软件需求和验收要求编制测试计划，制定需测试的测试项，制定测试策略及验收通过准则，并经过客户参与的计划评审。

**4．测试设计和测试用例设计**

根据《验收测试计划》和《项目验收准则》编制测试用例，并经过评审。

**5．测试环境搭建**

建立测试的硬件环境、软件环境等。（可在委托客户提供的环境中进行测试）

**6．测试实施**

测试并记录测试结果。

**7．测试结果分析**

根据验收通过准则分析测试结果，作出验收是否通过及测试评价。

**8．测试报告**

根据测试结果编制缺陷报告和验收测试报告，并提交给客户。

## 8.3　验收测试的常用策略

验收测试是部署软件之前的最后一个测试操作。验收测试的目的是确保软件准备就绪，并且可以让最终用户将其用于执行软件的既定功能和任务。实施验收测试的常用策略有三种，它们分别是：

（1）正式验收。

（2）非正式验收或 Alpha 测试。

（3）Beta 测试。

您选择的策略通常建立在合同需求、组织和公司标准以及应用领域的基础上。

### 8.3.1 正式验收测试

正式验收测试是一项管理严格的过程，它通常是系统测试的延续。计划和设计这些测试的周密和详细程度不亚于系统测试。选择的测试用例应该是系统测试中所执行测试用例的子集。不要偏离所选择的测试用例方向，这一点很重要。在很多组织中，正式验收测试是完全自动执行的。

对于系统测试，活动和工件是一样的。在某些组织中，开发组织（或其独立的测试小组）与最终用户组织的代表一起执行验收测试。在其他组织中，验收测试则完全由最终用户组织执行，或者由最终用户组织选择人员组成一个客观公正的小组来执行。

这种测试形式的优点如下：

- 要测试的功能和特性都是已知的。
- 测试的细节是已知的并且可以对其进行评测。
- 这种测试可以自动执行，支持回归测试。
- 可以对测试过程进行评测和监测。
- 可接受性标准是已知的。

缺点包括以下几点：

- 要求大量的资源和计划。
- 这些测试可能是系统测试的再次实施。
- 可能无法发现软件中由于主观原因造成的缺陷，这是因为你只查找预期要发现的缺陷。

### 8.3.2 非正式验收测试

在非正式验收测试中，执行测试过程的限定不像正式验收测试中那样严格。在此测试中，确定并记录要研究的功能和业务任务，但没有可以遵循的特定测试用例。测试内容由各测试员决定。这种验收测试方法不像正式验收测试那样组织有序，而且更为主观。

大多数情况下，非正式验收测试是由最终用户组织执行的。

这种测试形式的优点如下：

- 要测试的功能和特性都是已知的。
- 可以对测试过程进行评测和监测。
- 可接受性标准是已知的。
- 与正式验收测试相比，可以发现更多由于主观原因造成的缺陷。

缺点包括以下几点：

- 要求资源、计划和管理资源。
- 无法控制所使用的测试用例。
- 最终用户可能沿用系统工作的方式，并可能无法发现缺陷。
- 最终用户可能专注于比较新系统与遗留系统，而不是专注于查找缺陷。
- 用于验收测试的资源不受项目的控制，并且可能受到压缩。

### 8.3.3 Beta 测试

在三种验收测试策略中，Beta 测试需要的控制是最少的。在 Beta 测试中，采用的细节多少、数据和方法完全由各测试员决定。各测试员负责创建自己的环境、选择数据，并决定要研究的功能、特性或任务。各测试员负责确定自己对于系统当前状态的接受标准。

Beta 测试由最终用户实施，通常开发（或其他非最终用户）组织对其的管理很少或不进行管理。Beta 测试是所有验收测试策略中最主观的。

这种测试形式的优点如下：

- 测试由最终用户实施。
- 大量的潜在测试资源。
- 提高客户对参与人员的满意程度。
- 与正式或非正式验收测试相比，可以发现更多由于主观原因造成的缺陷。

缺点包括以下几点：

- 未对所有功能和特性进行测试。
- 测试流程难以评测。
- 最终用户可能沿用系统工作的方式，并可能没有发现或没有报告缺陷。
- 最终用户可能专注于比较新系统与遗留系统，而不是专注于查找缺陷。
- 用于验收测试的资源不受项目的控制，并且可能受到压缩。
- 可接受性标准是未知的。
- 需要更多辅助性资源来管理 Beta 测试员。

## 8.4 验收测试的总体思路

用户验收测试是软件开发结束后，用户对软件产品投入实际应用以前进行的最后一次质量检验活动。它要回答开发的软件产品是否符合预期的各项要求，以及用户能否接受的问题。由于它不只是检验软件某个方面的质量，而是要进行全面的质量检验，并且要决定软件是否合格，因此验收测试是一项严格的正式测试活动。需要根据事先制订的计划，进行软件配置评审、功能测试、性能测试等多方面检测。

用户验收测试可以分为两个大的部分：软件配置审核和可执行程序测试，其大致顺

序可分为：文档审核、源代码审核、配置脚本审核、测试程序或脚本审核、可执行程序测试。

要注意的是，在开发方将软件提交用户方进行验收测试之前，必须保证开发方本身已经对软件的各方面进行了足够的正式测试（当然，这里的“足够”，本身是很难准确定量的）。

用户在按照合同接收并清点开发方的提交物时（包括以前已经提交的），要查看开发方提供的各种审核报告和测试报告内容是否齐全，再加上平时对开发方工作情况的了解，基本可以初步判断开发方是否已经进行了足够的正式测试。

用户验收测试的每一个相对独立的部分，都应该有目标（本步骤的目的）、启动标准（着手本步骤必须满足的条件）、活动（构成本步骤的具体活动）、完成标准（完成本步骤要满足的条件）和度量（应该收集的产品与过程数据）。在实际验收测试过程中，收集度量数据，不是一件容易的事情。

### 8.4.1 软件配置审核

对于一个外包的软件项目而言，软件承包方通常要提供如下相关的软件配置内容。

- 可执行程序、源程序、配置脚本、测试程序或脚本。
- 主要的开发类文档：《需求分析说明书》、《概要设计说明书》、《详细设计说明书》、《数据库设计说明书》、《测试计划》、《测试报告》、《程序维护手册》、《程序员开发手册》、《用户操作手册》、《项目总结报告》。
- 主要的管理类文档：《项目计划书》、《质量控制计划》、《配置管理计划》、《用户培训计划》、《质量总结报告》、《评审报告》、《会议记录》、《开发进度月报》。

在开发类文档中，容易被忽视的文档有《程序维护手册》和《程序员开发手册》。《程序维护手册》的主要内容包括系统说明（包括程序说明）、操作环境、维护过程、源代码清单等，编写目的是为将来的维护、修改和再次开发工作提供有用的技术信息。《程序员开发手册》的主要内容包括系统目标、开发环境使用说明、测试环境使用说明、编码规范及相应的流程等，实际上就是程序员的培训手册。

不同大小的项目，都必须具备上述的文档内容，只是可以根据实际情况进行重新组织。对上述的提交物，最好在合同中规定阶段提交的时机，以免发生纠纷。

通常，正式的审核过程分为 5 个步骤：计划、预备会议（可选）、准备阶段、审核会议和问题追踪。预备会议是对审核内容进行介绍并讨论。准备阶段就是各责任人事先审核并记录发现的问题。审核会议是最终确定工作产品中包含的错误和缺陷。

审核要达到的基本目标是：根据共同制定的审核表，尽可能地发现被审核内容中存在的问题，并最终得到解决。在根据相应的审核表进行文档审核和源代码审核时，还要注意文档与源代码的一致性。

在实际的验收测试执行过程中，常常会发现文档审核是最难的工作，一方面由于市

场需求等方面的压力使这项工作常常被弱化或推迟，造成持续时间变长，加大文档审核的难度；另一方面，文档审核中不易把握的地方非常多，每个项目都有一些特别的地方，而且也很难找到可用的参考资料。

### 8.4.2　可执行程序的测试

在文档审核、源代码审核、配置脚本审核、测试程序或脚本审核都顺利完成，就可以进行验收测试的最后一个步骤——可执行程序的测试，它包括功能、性能等方面的测试，每种测试也都包括目标、启动标准、活动、完成标准和度量等 5 部分。

要注意的是不能直接使用开发方提供的可执行程序用于测试，而要按照开发方提供的编译步骤，从源代码重新生成可执行程序。

在真正进行用户验收测试之前一般应该已经完成了以下工作（也可以根据实际情况有选择地采用或增加）：

- 软件开发已经完成，并全部解决了已知的软件缺陷。
- 验收测试计划已经过评审并批准，并且置于文档控制之下。
- 对软件需求说明书的审查已经完成。
- 对概要设计、详细设计的审查已经完成。
- 对所有关键模块的代码审查已经完成。
- 对单元、集成、系统测试计划和报告的审查已经完成。
- 所有的测试脚本已完成，并至少执行过一次，且通过评审。
- 使用配置管理工具且代码置于配置控制之下。
- 软件问题处理流程已经就绪。
- 已经制定、评审并批准验收测试完成标准。

具体的测试内容通常可以包括安装（升级）、启动与关机、功能测试（正例、重要算法、边界、时序、反例、错误处理）、性能测试（正常的负载、容量变化）、压力测试（临界的负载、容量变化）、配置测试、平台测试、安全性测试、恢复测试（在出现掉电、硬件故障或切换、网络故障等情况时，系统是否能够正常运行）、可靠性测试等。

性能测试和压力测试一般情况下是在一起进行，通常还需要辅助工具的支持。在进行性能测试和压力测试时，测试范围必须限定在那些使用频度高的和时间要求苛刻的软件功能子集中。由于开发方已经事先进行过性能测试和压力测试，因此可以直接使用开发方的辅助工具。也可以通过购买或自己开发来获得辅助工具。具体的测试方法可以参考相关的软件工程书籍。

如果执行了所有的测试案例、测试程序或脚本，用户验收测试中发现的所有软件问题都已解决，而且所有的软件配置均已更新和审核，可以反映出软件在用户验收测试中所发生的变化，用户验收测试就完成了。

## 8.5 验收测试报告

验收测试是 QA 在整个产品测试中的最后一个环节，完成并通过验收测试后我们需要提交验收测试报告，有时也称为发布报告（Release Report）。在报告中要综合分析各阶段所有的测试内容，有充分的信心保证产品的质量，并指出可能存在的问题。当然没有 Bug 的软件是不存在的，我们不能宣称找出并修正了软件中的所有错误和缺陷，有时迫于市场压力和时间上的考虑，我们会允许即将发布的软件中存在部分级别较低、对用户影响不大的缺陷。

# 第 9 章　软件测试管理及自动化测试基础

**学习目标：**

1．了解什么是自动化测试

2．熟悉软件自动化测试

3．了解自动化测试工具的分类

## 9.1　软件测试自动化基础

软件测试是一项艰苦的工作，需要投入大量的时间和精力，据统计，软件测试会占用整个开发时间的 40%。一些可靠性要求非常高的软件，测试时间甚至占到总开发时间的 60%。但是软件测试具有一定的重复性，软件在发布之前要进行几轮测试。在测试后期所进行的回归测试中大部分测试工作是重复的，回归测试就是要验证已经实现的大部分功能。这种情况下，代码修改很少，针对代码变化所做的测试相对较少。而为了覆盖代码改动所造成的影响需要进行大量的测试，虽然这种测试找到软件缺陷的可能性小，效率比较低，但又是必要的。此后，软件不断升级，所要做的测试重复性也很高，所有这些因素驱动着软件测试自动化的产生和发展。

### 9.1.1　自动化测试的引入

软件测试作为保证软件质量和可靠性的关键技术，正日益受到广泛的重视，但随着软件工程的规模越来越大，客户对软件的质量要求越来越高，测试的工作量也越来越大。如何进行测试，如何提高测试的质量和效率，从而确保软件产品的质量和可靠性，就成了许多人深感困扰的问题。

目前，企业级应用系统越来越多，这些系统可能包括 ERP 系统、CRM 系统等。这些系统在发布之前或升级之后都要经过测试，确保主要功能都能正常运行，错误最少。如何有效地测试不断升级和不断更换应用环境的应用系统，是每个公司都会面临的问题。如果时间或资源有限，这个问题会更加棘手。人工测试的工作量太大，同时还需要额外的时间来培训测试人员等。为了确保那些复杂的企业级应用在不同环境下都能可靠地运行，需要一个能简单操作的测试工具来自动完成应用程序的功能性测试。

同时，目前企业的网络应用环境都必须支持大量用户和不同的软硬件应用环境。难以预知的用户负载和越来越复杂的应用环境使公司时时担心会发生用户响应速度过慢、系统崩溃等问题。这些都不可避免地导致公司收益的损失。为了在终端用户正式使用前，

对应用系统各个环节的质量、可靠性和可扩展性进行测试和评价，就需要适用于不同体系架构的自动负载压力测试工具，以预测系统行为并为系统优化提供依据。

总之，为了更加快速、有效地对软件进行测试，提高软件产品的质量，我们必然会利用测试工具，也必然会引入自动化测试。

### 9.1.2 自动化测试的含义

自动化测试是相对于手工测试而存在的，主要是使用软件工具来代替手工进行的一系列动作，具有良好的可操作性、可重复性和高效率等特点。自动化测试的目的是减轻手工测试的工作量，以达到节约资源（包括人力、物力等），保证软件质量，缩短测试周期的效果，是软件测试中提高测试效率、覆盖率和可靠性的重要测试手段。也可以说，自动化测试是软件测试不可分割的一部分。

自动化测试将毫无差错地以同一方式多次运行同一测试。但是自动化测试不会执行与脚本编写的内容不一样的行为。正因为如此，自动化测试通常被看成为一系列的回归测试，只能捕获被引入原来工作代码的缺陷。不过事情也会出现例外，例如大型数据数组循环输入。但是可以肯定自动化测试大都属于回归测试的范畴。

### 9.1.3 自动化测试的意义

测试人员在进行手工测试时，具有创造性，可以举一反三，从一个测试用例想到另外一个测试用例，特别是可以考虑到测试用例没有覆盖的一些特殊的或边界的情况。同时，对于那些复杂的逻辑判断、界面是否友好，手工测试具有明显的优势。但是手工测试在某些测试方面，可能还存在着一定的局限性，例如，通过手工测试无法做到覆盖所有代码路径；简单的功能性测试用例在每一轮测试中都不能少，而且具有一定的机械性、重复性，其工作量往往较大，却无法体现手工测试的优越性；在系统负载、性能测试时，需要模拟大量数据或大量并发用户等各种应用场合，很难通过手工测试来进行。

由于手工测试的局限性，软件测试借助软件工具向自动化测试方向发展就显得极为必要。通过自动化测试，可以解决上述手工测试的局限性，带来以下好处。

**1．提高测试效率**

手工测试是一个劳动密集型的工作，并且容易出错。引入自动测试能够用更有效、可重复的自动化测试环境代替烦琐的手工测试活动，而且能在更少的时间内完成更多的测试工作，从而提高了测试工程师的工作效率。

**2．降低对软件新版本进行回归测试的开销**

对于现代软件的迭代增量开发，每一个新版本大部分功能和界面都和上一个版本相似或完全相同，这时要对新版本再次进行已有的测试，这部分工作多为重复工作，特别适合使用自动化测试来完成，从而减小回归测试的开销。

**3．完成手工测试不能或难以完成的测试**

对于一些非功能性方面的测试，如压力测试、并发测试、大数据量测试、崩溃性测试等，这些测试用手工测试是很难，甚至是不可能完成的。但自动化测试能方便地执行这些测试，比如并发测试，使用自动化测试工具就可以模拟来自多方的并发操作。

**4．具有一致性和可重复性**

由于每次自动化测试运行的脚本是相同的，所以可以进行重复的测试，使得每次执行的测试具有一致性，手工测试则很难做到这点。

**5．更好地利用资源**

将烦琐的测试任务自动化，可以使测试人员解脱出来，将精力更多地投入到测试案例的设计和必要的手工测试当中。并且理想的自动化测试能够按计划完全自动地运行，使得完全可以利用非工作时间执行自动测试。

**6．降低风险，增加软件信任度**

自动化测试能通过较少的开销获得更彻底的测试效果，从而更好地提高了软件产品的质量。

## 9.1.4　自动化测试的优势

自动化测试能够替代大量手工测试工作，避免重复测试，同时，它还能够完成大量手工无法完成的测试工作，如并发用户测试、大数据量测试、长时间运行可靠性测试等，概括地说，自动化测试具有如下优点。

（1）提高测试质量：软件开发的过程就是一个持续集成和改进的过程，而每一次修改都有可能产生错误。因此，当软件产品的一部分，或者全部，或者应用环境被修改时都需要对软件产品重新进行测试，其目的是验证修改后的系统或者产品的质量是否符合规格说明。例如，对于产品型的软件，每发布一个新的版本，其中大部分功能和界面都和上一个版本相似或完全相同，这部分功能特别适合于自动化测试，由于自动测试工具提供了简便的回归测试，能以便利的方式验证是否有新的错误进入软件产品，既节省了重复手工输入的工作量，保证了测试案例的一致性，避免了人为因素，也可以使测试达到测试每个质量特性的目的，从而提高软件测试的质量。

（2）提高测试效率，缩短测试工作时间：软件系统的规模越来越大，功能点越来越多，达到几千个上万个，人工测试非常耗时和烦琐，这样必然会导致测试效率低下，而自动化测试工具可以较好地执行这些频繁的测试任务。在充分并合理地使用了测试工具以后，可以减轻测试工程师的手工测试工作，同时，测试工具还可以把控制和管理引入整个测试过程，能够保证测试的进度。

（3）提高测试覆盖率：通过自动化测试工具的录制回放及数据驱动来测试功能，可以提高测试覆盖率。通过测试工具的辅助分析功能，可以提高测试的深度。

（4）执行手工测试不能完成的测试任务：有些非功能性方面的测试，如压力测试、

负载测试、大数据量测试、崩溃性测试等，人工测试是不可能实现的。例如，找若干台电脑和同样数目的操作人员在同一时刻进行操作，然后拿秒表记录下反应时间，这样的手工作坊式的测试方法不切实际且无法捕捉程序内部变化情况。

（5）更好地重现软件缺陷的能力：自动化测试具有更好的一致性和可重复性，由于每次自动化测试运行的脚本是相同的，所以每次执行的测试具有一致性，人是很难做到的。由于自动化测试的一致性，很容易发现被测软件的任何改变。

（6）更好地利用资源：理想的自动化测试能够按计划完全自动地运行，在开发人员和测试人员不可能实行三班倒的情况下，自动化测试可以胜任这个任务，例如，完全可以在周末或者晚上执行测试。这样充分地利用资源，也避免了开发和测试之间的冲突。

（7）增进测试人员与开发人员之间的合作伙伴关系：测试工程师为了更好地使用自动化测试工具，需要对开发技术有深入的理解和实践，因此测试工程师也有了与开发工程师更多、更平等地交流的机会，自动化测试为测试工程师与程序开发人员协同工作提供了一种便利的手段，双方将有更多的合作与尊重。

测试工具能够提高软件质量，改进测试过程，因此在许多公司中得到了广泛应用，由于自动化测试工具自身的特点，为达到较高的投资回报率，在以下项目和环境中更适合使用自动化测试工具。

（1）需要反复进行的工作。在持续修改软件功能的项目中，对功能的测试需要反复进行，人工测试工作量极大。功能性测试工具能够自动进行重复性的工作，验证软件的修改是否引入了新的缺陷，旧的缺陷是否已经修改。减少人工测试的工作量。

（2）负载压力测试。负载压力测试需要模拟大量并发用户和大数据量，这样的测试用手工不能完成或不能很好地完成，而自动化测试工具则可以很好地解决这个问题，在测试脚本运行过程中也不需要人工干预，能够充分利用非工作时间。

（3）公司有大量的测试人员和开发人员，他们合作完成一个产品，那么如何在产品的生命周期中进行有效管理和合作，借助于自动化的测试管理工具，会取得事半功倍的效果。

（4）如果需要进行测试系统后台或者内部的性能特性，进而进行故障定位和性能调优，自动化测试工具会是一个不错的选择。

### 9.1.5 自动化测试的局限性

当然，自动化测试也并非万能，它所完成的测试功能也是有限的，不可能也没有必要取代手工测试来完成所有的测试任务。以下几点是自动化测试的不足。

（1）软件自动化测试可能降低测试的效率。当测试人员只需要进行很少量的测试，而且这种测试在以后的重用性很低时，花大量的精力和时间去进行自动化的结果往往是得不偿失。因为自动化的收益一般要在很多次重复使用中才能体现出来。

（2）测试人员期望自动测试发现大量的错误。测试首次运行时，可能发现大量错误。

但当进行过多次测试后，发现错误的几率会相对较小，除非对软件进行了修改或在不同的环境下运行。

（3）如果缺乏测试经验，测试的组织差、文档少或不一致，则自动化测试的效果比较差。

（4）技术问题。毫无疑问商用软件自动测试工具是软件产品。作为第三方的技术产品，如果不具备解决问题的能力和技术支持或者产品适应环境变化的能力不强，将使得软件自动化工具的作用大大降低。

因此，对软件自动化测试应该有正确的认识，它并不能完全代替手工测试。不要期望仅仅通过自动化测试就能提高测试的质量，如果测试人员缺少测试的技能，那么测试也可能会失败。

### 9.1.6　测试工具

测试工具可以从以下两个不同的方面去分类。

根据测试方法不同，分为白盒测试工具和黑盒测试工具。

根据测试的对象和目的，分为单元测试工具、功能测试工具、负载测试工具、性能测试工具和测试管理工具等。

**1．白盒测试工具**

白盒测试工具是针对程序代码、程序结构、对象属性、类层次等进行测试，测试中发现的缺陷可以定位到代码行、对象或变量级。根据测试工具原理的不同，又可以分为静态测试工具和动态测试工具。

静态测试工具对代码进行语法扫描，找出不符合编码规范的地方，根据某种质量模型评价代码的质量，生成系统的调用关系图等。它直接对代码进行分析，不需要运行代码，也不需要对代码编译链接、生成可执行文件。

动态测试工具与静态测试工具不同，需要实际运行被测系统，并设置断点，向代码生成的可执行文件中插入一些监测代码，掌握断点这一时刻程序运行数据（对象属性、变量的值等）。单元测试工具多属于白盒测试工具。

**2．黑盒测试工具**

黑盒测试工具适用于系统功能测试和性能测试，包括功能测试工具、负载测试工具、性能测试工具等。黑盒测试工具的一般原理是利用脚本的录制（Record）/回放（Playback），模拟用户的操作，然后将被测系统的输出记录下来同预先给定的标准结果比较。黑盒测试工具可以大大减轻黑盒测试的工作量，在迭代开发的过程中，能够很好地进行回归测试。

**3．其他测试工具**

在上述两类测试工具之外还有测试管理工具，这类工具负责对测试计划、测试用例、测试实施进行管理，对产品缺陷跟踪管理、产品特性管理等。除了上述的测试工具外，

还有一些专用的测试工具，例如，针对数据库测试的 TestBytes，对应用性能进行优化的 EcoScope 等工具。

## 9.2 软件测试管理

随着计算机硬件成本的不断下降，软件在整个计算机系统的成本占有越来越高的比例，如何提高软件质量是整个计算机软件行业的重大课题。软件测试作为软件开发的一个重要环节，越来越受重视。为了尽可能多地找出程序中的错误，保证软件产品的质量，就需要对软件测试进行有效的管理，确保测试工作顺利进行。

实践证明，对软件进行测试管理可及早发现错误，避免大规模返工，降低软件开发费用。为确保最终软件质量符合要求，必须进行测试与管理。对于不同企业的不同类产品、不同企业的同一类产品或同一企业的不同类产品，其各阶段结果的形式与内容都会有很大的不同。所以，对于软件测试管理除了要考虑测试管理开始的时间、测试管理的执行者、测试管理技术如何有助于防止错误的发生、测试管理活动如何被集成到软件过程的模型中之外，还必须在测试之前，制订详细的测试管理计划，充分实现软件测试管理的主要功能，缩短测试管理的周期。

### 9.2.1 软件测试管理计划

一个成功的测试开始于一个全面的测试管理计划。因此，在每次测试之前应做好详细的测试管理计划。

首先，应该了解被测对象的基本信息，选择测试的标准级别，明确测试管理计划标识和测试管理项。在定义了被测对象的测试管理目标、范围后必须确定测试管理所使用的方法，提供技术性的测试管理策略和测试管理过程。在测试管理计划中，管理者应该全面了解被测试对象的系统方法、语言特征、结构特点、操作方法和特殊需求等，以便确定必要的测试环境，包括测试硬件、软件及测试环境的建立等。而且，在测试管理计划中还应该制订一份详细的进度计划，如测试管理的开始段、中间段、结束段及测试管理过程每个部分的负责人等。

其次，由于任何一个软件不可能没有缺陷、系统运行时不出现故障，所以在测试管理计划中还必须考虑到一些意外情况，也就是说，当问题发生时应如何处理。因为测试管理具有一定难度，所以对测试管理者应进行必要的测试设计、工具、环境等的培训。

最后，还必须确定认可和审议测试管理计划的负责人员。

### 9.2.2 软件测试管理主要功能

**1．测试控制对象的编辑和管理**

测试控制对象包括测试方案、测试案例、各案例的具体测试步骤、问题报告、测试结果报告等，该部分主要是为各测试阶段的控制对象提供一个完善的编辑和管理环境。

**2．测试流程的控制和管理**

测试流程的控制和管理是基于科学的流程和具体的规范来实现的，并利用该流程和规范，严格约束和控制整个产品的测试周期，以确保产品的质量。整个过程避免了测试人员和开发设计人员之间面对面的交流，减少了以往测试和开发之间难免的摩擦和矛盾，提高了工作效率。

**3．统计分析和决策支持**

在系统建立的测试数据库的基础上，进行合理的统计分析和数据挖掘，例如根据问题分布的模块、问题所属的性质、问题的解决情况等方面的统计分析使项目管理者全面了解产品开发的进度、产品开发的质量、产品开发中问题的聚集，为决策管理提供支持。例如，设计人员在遇到问题时可以到案例库中查找类似问题的解决办法等。

### 9.2.3　软件测试管理实施

任何程序，无论大小，都可能会有错误发生。每一个新版本都需要进行新特性的测试和其他特性的一些回归测试。所以软件测试管理具有周期性。

测试管理人员在接受一个测试管理任务后，除了要制订周密的测试管理计划。还要进行测试方案管理；并且对测试人员所做的测试活动予以记录，做好测试流程的管理。同时，对发现的缺陷予以标识，一方面反馈给提交测试的人员；另一方面将存在的问题和缺陷存入案例库，直至测试通过。

软件测试是一个完整的体系，主要由测试规划、测试设计、测试实施、资源管理等相互关联、相互作用的过程构成。软件测试管理系统可以对各过程进行全面控制，具体的实现过程如下。

（1）按照国际质量管理标准，建立适合本公司的软件测试管理体系，以提高公司开发的软件质量，并降低软件开发及维护的成本。

（2）建立、监测和分析软件测试过程，以有效地控制、管理和改进软件测试过程。监测软件质量，从而确定交付或发布软件的时间。

（3）制订合理的软件测试管理计划，设计有效的测试案例集，以尽可能发现软件缺陷，并组织、管理和应用庞大的测试案例集。

（4）在软件测试管理过程中，管理者、程序员、测试员（含有关客户人员）协同工作，及时解决发现的软件问题。

（5）对于软件测试中发现的大量的软件缺陷，进行合理的分类以分清轻重缓急。同时进行原因分析，并做好相应的记录、跟踪和管理工作。

（6）建立一套完整的文档资料管理体系。因为软件测试管理很大程度上是通过对文档资料的管理来实现的。软件测试每个阶段的文档资料都是以后阶段的基础，又是对前面阶段的复审。

### 9.2.4 软件测试管理工具简介

在软件测试管理周期中，为了便于对制定测试方案、编写测试案例和测试步骤等各个阶段进行有效的控制和管理，为了提高软件开发和产品测试的管理水平，保证软件产品质量，软件测试管理工具是非常重要的手段。在此介绍以下一些比较流行的软件测试管理工具。

**1. Mercury Quality Center（TestDirector）**

这是业界第一个基于 Web 的测试管理系统，它可以在您公司组织内进行全球范围内测试的协调。通过在一个整体的应用系统中提供并且集成了测试需求管理、测试计划、测试日程控制以及测试执行和错误跟踪等功能，TestDirector 极大地加速了测试过程。网址：http://www.mercury.com。

**2. Rational TestManager**

这是针对测试活动管理、执行和报告的中央控制台。它是为可扩展性而构建的，支持的范围从纯人工测试方法到各种自动化范型（包括单元测试、功能回归测试和性能测试）。Rational TestManager 可以由项目团队的所有成员访问，确保了测试覆盖信息、缺陷倾势和应用程序准备状态的高度可见性。网址：http://www-306.ibm.com/software/rational/。

**3. QA Director**

分布式的测试能力和多平台支持，能够使开发和测试团队跨越多个环境控制测试活动，QADirector 允许开发人员、测试人员和 QA 管理人员共享测试资产，测试过程和测试结果、当前的和历史的信息。从而为客户提供了最完全彻底的、一致的测试。网址：http://www.compuware.com。

**4. SilkCentral Test Manager（SilkPlan Pro）**

这是一个完整的测试管理软件。用于测试的计划、文档和各种测试行为的管理。它提供对人工测试和自动测试的基于过程的分析、设计和管理功能，此外，还提供了基于 Web 的自动测试功能。这使得 SilkPlan Pro 成为 Segue Silk 测试家族中的重要成员和用于监测的解决方案。在软件开发的过程中，SilkPlan Pro 可以使测试过程自动化，节省时间，同时帮助你回答重要的业务应用面临的关键问题。网址：http://www.segue.com。

这些软件测试管理工具可以为企业商业系统提供全面的、综合的测试管理解决方案，并可以控制和管理所有的测试工作来确保测试是一个有组织的、规范文档化的和全面的测试活动。

## 9.3 选择合适的自动化测试工具

### 9.3.1 自动化测试工具分类

自动化测试工具可以减少测试工作量，提高测试工作效率，但首先是能够选择一个

合适的且满足企业信息系统工程环境的自动化测试工具，因为不同的测试工具，其面向的测试对象是不一样的。按照测试工具的主要用途和应用领域，可以将自动化测试工具分为以下几类。

（1）**负载压力测试工具**：这类测试工具的主要目的都是为了度量应用系统的可扩展性和性能，是一种预测系统行为和性能的自动化测试工具。它们通过模拟成百上千直至上万用户并发执行关键业务，而完成对应用程序的测试，在实施并发负载过程中通过实时性能监测来确认和查找问题，并针对所发现问题对系统性能进行优化，确保应用的成功部署。负载压力测试工具能够对整个企业架构进行测试，通过这些测试，企业能最大限度地缩短测试时间，优化性能和加速应用系统的发布周期。这类工具的主要代表有LoadRunner、QALoad、SILK PERFORMA V 和 E-Test Suite 等。

（2）**功能测试工具**：通过自动录制、检测和回放用户的应用操作，将被测系统的输出记录同预先给定的标准结果进行比较，功能测试工具能够有效地帮助测试人员对复杂的企业级应用的不同发布版本的功能进行测试，提高测试人员的工作效率和质量。其主要目的是用于检测应用程序是否能够达到预期的功能并正常运行。功能测试工具可以大大减轻黑盒测试的工作量，在迭代开发的过程中，能够很好地进行回归测试。这类工具的主要代表有 WinRunner、QTP、QARun 等。

（3）**白盒测试工具**：白盒测试工具一般是针对代码进行测试，测试中发现的缺陷可以定位到代码级，根据测试工具原理的不同，又可以分为静态测试工具和动态测试工具。静态测试工具直接对代码进行分析，不需要运行代码，也不需要对代码编译链接和生成可执行文件。静态测试工具一般是对代码进行语法扫描，找出不符合编码规范的地方，根据某种质量模型评价代码的质量，生成系统的调用关系图等。静态测试工具的代表有Logiscope 软件和 PRQA 软件。动态测试工具与静态测试工具不同，动态测试工具一般采用“插桩”的方式，向代码生成的可执行文件中插入一些监测代码，用来统计程序运行时的数据。其与静态测试工具最大的不同就是动态测试工具要求被测系统实际运行。动态测试工具的代表有 DevPartner、Rational Purify 系列等。

（4）**网络测试工具**：这类工具主要包括网络故障定位工具、网络性能监测工具、网络仿真模拟工具等。它们分析分布式应用性能，关注应用、网络和其他元素（如服务器）内部的交互式活动，以便使网络管理员能够了解网络不同位置和不同活动之间应用的行为。你可以用它在交易执行过程中、Web 查找和检索中或在日常数据库上传/下载中跟踪应用行为。它可在会话级、代码级，甚至在帧级和包级观察应用的行为过程，并深入代码内部的结构，解析有问题的会话。

（5）**测试管理工具**：测试管理工具用于对测试进行管理。一般而言，测试管理工具对测试需求、测试计划、测试用例、测试实施进行管理，并且测试管理工具还包括对缺陷的跟踪管理。测试管理工具能让测试人员、开发人员或其他的 IT 人员通过一个中央数据仓库，在不同的地方就能交互信息。测试管理工具将测试过程流水化，从测试需求管

理到测试计划、测试日程安排、测试执行到出错后的错误跟踪，实现了全过程的自动化管理。测试管理工具的代表有 TestDirector、TestManger、TrackRecord 等。

（6）**测试辅助工具**：这些工具本身并不执行测试，例如它们可以生成测试数据，为测试提供数据准备等。

## 9.3.2　自动化测试应用策略

随着软件测试地位的逐步提高，测试的重要性逐步显现，测试工具的应用已经成为了普遍的趋势。总的来说，测试工具的应用可以提高测试的质量、测试的效率。但是在选择和使用测试工具的时候，我们也应该看到，在测试过程中，并不是所有的测试工具都适合我们使用，同时，有了测试工具并且会使用测试工具并不等于测试工具真正能在测试中发挥作用，因此，如何确定自动化测试策略显得至关重要。应用测试工具的目的很明确，一般而言，在测试过程中应用测试工具主要有以下几个目的。

（1）提高测试质量。

（2）减少测试过程中的重复劳动。

（3）实现测试自动化，解决手工测试不能解决的问题。

为了更好地达到测试目的，在信息系统中应用自动化测试需要考虑以下问题。

第一选择合适的自动化测试工具：面对众多不同用途的测试工具，如何正确地选择合适的测试工具，是能否正常实施自动化测试的前提，我们在选用工具的时候，建议从以下 3 个方面来权衡。

（1）功能。功能当然是我们最关注的内容，选择一个测试工具首先就是看它提供的功能。当然，这并不是说测试工具提供的功能越多越好，在实际的选择过程中，适用才是根本。“钱要花在刀刃上”，为不需要的功能花费金钱是不明智的行为。事实上，目前市面上同类的软件测试工具之间的基本功能都是大同小异的，各种软件提供的功能也大致相同，只不过有不同的侧重点。例如，同为白盒测试工具的 Logiscope 和 PRQA 软件，它们提供的基本功能大致相同，只是在编码规则、编码规则的定制、采用的代码质量标准方面有不同。除了基本的功能之外，以下的功能需求也可以作为选择测试工具的参考。

- 报表功能：测试工具生成的结果最终要由测试人员进行解释，而且，查看最终报告的人员不一定对测试很熟悉，因此，测试工具能否生成结果报表，能够以什么形势提供报表是需要考虑的因素（标准符号有些混乱）。
- 测试工具的集成能力：引入测试工具是一个长期的过程，应该是伴随着测试过程改进而进行的一个持续的过程。因此，测试工具的集成能力也是必须考虑的因素，这里的集成包括两个方面的意思，首先，测试工具能否和开发工具进行良好的集成；其次，测试工具能否和其他测试工具进行良好的集成。
- 操作系统和开发工具的兼容性。测试工具可否跨平台，是否适用于公司目前使用的开发工具，这些问题也是在选择一个测试工具时必须考虑的问题。

（2）价格。除了功能之外，价格就应该是最重要的因素了，作为一个测试工程师，在选择购买测试工具时，应该具有成本意识，必须利用有限的资金满足企业对测试工具的大多数需求。

（3）测试工具的长期投资考虑。测试工具引入的目的是测试自动化，引入工具需要考虑工具的连续性和一致性，也就是说，对测试工具的选择必须有一个全盘的考虑，分阶段、逐步的引入测试工具。

第二确定测试工具的应用时机：购买了测试工具以后，如何让测试工具真正发挥作用，是应用自动化测试的关键。任何测试工具都有其应用范围，也许我们具备不同的测试工具，那么在不同的软件工程阶段，我们应该有计划地去使用相应的测试工具，并将测试工具的使用明确定义进公司的开发流程。例如，在单元测试阶段，我们应该重点采用白盒测试工具，当软件产品的功能以及用户界面基本确定和逐步实现后，则可以考虑开始使用功能测试工具。集成测试阶段，则可以引入负载压力测试工具，对系统可能承受的负载压力进行测试与评估，并辅以相应的资源，使用监控工具进行故障定位等。

第三确定测试重点：对于一些测试项目，尤其是在测试时间有限的情况下，比如，执行一次性能测试，我们必须能够确定被测项目的主要应用和关键步骤，应该对那些质量要求较高并且风险大的部分进行重点测试，例如在金融领域，对于那些每天管理数百万、数千万人民币流动的系统，需要特别对其硬件、软件的安全可靠性、可用性进行测试。

第四确定测试目标和指标：针对不同的软件，其软件质量要求的等级和目标是不一样的，通过测试工具可以更好地验证系统设计是否达到了预期目标，因此，在正式开始测试前，我们应该能够清楚地了解测试预期目标。

第五充分利用测试工具的优势：每个测试工具都有自己独特的实现技术，对于同一个测试项目，测试工具可能也提供了多种测试方案供选择，比如脚本录制过程中协议的选择，回放过程中用户并发模拟机制和方式的选择等，只有充分利用了测试工具提供的这些技术，才可能更好、更真实地测试应用系统的实际质量。

第六加强对测试工程师的技能培训，测试工具的使用者必须对测试工具非常了解：在这方面，有效的培训是必不可少的。测试工具的培训是一个长期的过程，不是通过一两次讲课的形式就能达到良好的效果的。而且，在实际使用测试工具的过程中，测试工具的使用者可能还存在着这样那样的问题，这也需要有专家负责解决，否则的话，对于测试工具使用者的积极性将造成很大的打击。

### 9.3.3　功能自动化测试

**1．概述**

传统的手工测试是测试人员执行测试用例，然后将测试结果和预期结果相比较并记录测试结果。然而，随着软件工程的规模越来越大，软件产品的功能越来越复杂，同时，

软件的更新换代也越来越频繁，软件测试部门的工作难度越来越大，手工测试已经跟不上这种发展趋势了。

功能自动化测试工具可以帮助测试工程师自动处理测试开发到测试执行的整个过程中的问题。你可以创建可修改且可复用的测试脚本，甚至可以在下班后让计算机自动执行脚本，从而减少劳动量，提高测试效率。

功能测试自动化工具的主要功能，就是为了确保应用能够按照预期设计执行而将业务处理过程记录到测试脚本中。当应用被开发完成或应用升级时，测试工具支持测试脚本的编辑、扩展、执行和报告测试结果，并且保证测试脚本的可重复使用，贯穿于应用的整个生命周期。

当一个应用开发完毕后，程序界面基本定型，这个时候，针对该应用的自动测试应该展开。自动测试的引入，大大提高了测试的效率和测试的准确性，而且测试人员一次设计的脚本，可以在软件生命周期的各个阶段重复使用，尤其在软件交付后，随着企业的发展，你的应用就会随之在数量和范围上增长。为了满足业务的需求，应用的改变会很频繁，对于这些需求，将可以通过小范围修改测试工具录制的脚本来完成。对于功能测试工具的使用，比较重要的是测试规划问题。如何规划一次录制，使它具有良好的可扩展性、重用性，整个脚本能够有清晰的层次和最大限度地适应以后程序的修改，这些也是在实际工程中用户最普遍遇到的问题，它的实施就需要有经验的软件测试人员介入并结合应用来进行具体分析。

**2．测试原理**

功能自动化测试工具基本上都是采取录制回放的方式来模拟用户的实际操作。当你在软件操作中点击图形用户界面上的对象时，测试工具会用一种类 C 或者其他的脚本语言（TSL）生成一个测试脚本，该脚本记录了你的操作过程，然后测试工具就可以回放刚才的操作过程。当然你也可以手工编程生成这个脚本。通常情况下，测试工具采取两种录制模式。

（1）环境判断模式。

这种模式根据你选取的图形用户界面对象（如窗体、清单、按钮等），把你对软件的操作动作录制下来，并忽略这些对象在屏幕上的物理位置。每一次你对被测软件进行操作，测试脚本语言会记录并描述你选取的对象和你的操作动作。当你进行录制时，测试工具会对你选取的每个对象做唯一描述并写入相应的文件中。当软件用户界面发生变化时，你只需更新特定的对象记录文件。这样一来，环境判断模式的测试脚本将非常容易地被重复使用。执行测试只需要回放测试脚本。回放时，测试工具从指定文件中读取对象描述，并在被测软件中查找符合这些描述的对象并模拟用户使用鼠标选取该对象、用键盘输入数据的操作。

（2）模拟模式。

这种模式记录鼠标点击、键盘输入和鼠标在二维平面上（X 轴和 Y 轴）的精确运动轨迹。执行测试时，测试工具让鼠标根据轨迹运动。这种模式对于那些需要追踪鼠标运动的测试非常有用。不论测试工具采用的是哪种录制模式，通常情况下，其实施测试必须经历的几个操作步骤如下。

- 创建脚本：你可以通过录制、编程或两者同用的方式创建测试脚本。测试工具可以自动记录你的操作并生成所需的脚本代码，你还可以直接修改测试脚本以满足各种复杂测试的需求。录制测试时，在需要检查软件反应的地方插入检查点（Checkpoint）。你可以插入检查点来检查 GUI 对象、位图（Bitmap）和数据库。在这个过程中，测试工具会自动捕捉数据，并将该数据作为期望结果储存下来。例如，在创建测试时，可以设定哪些数据库表和记录需要检测，在测试运行时，测试程序就会自动核对数据库内的实际数值和预期的数值是否一致。
- 调试脚本：脚本录制或编辑结束后，你可以先在调试模式下运行脚本。并可以设置断点（Breakpoint）来监测变量，控制对象识别和隔离错误。
- 执行测试：脚本调试结束后，便可以在检验模式下测试被测软件。运行测试时，测试工具会自动操作应用程序，就像一个真实的用户根据业务流程执行着每一步的操作。此时，测试工具在运行脚本过程中如果遇到了检查点，就把当前数据和事先捕捉并保存的期望值进行比较。如果发现有不符合，就记录下来作为测试结果。在具体的测试过程中，为了全面地测试一个应用程序，需要使用不同类型的数据来测试。一般情况下，测试工具都能提供动态数据处理及参数化技术，可以用参数去代替定值，从而真实地反映多个用户行为。以一个订单输入的流程为例，你可能希望把订单号或客户名称作为可变栏，用多套数据进行测试。使用数据驱动向导，你可以选择订单号或客户名称用数据表格文件中的哪个栏目的数据替换。你可以把订单号或客户名称输入数据表格文件，或从其他表格和数据库中导入。数据驱动测试不仅节省了时间和资源，又提高了应用的测试覆盖率。
- 结果分析：每次测试结束，测试工具都会把测试情况显示在测试结果报告中。测试结果报告会详细描述测试执行过程中发生的所有主要事件，如检查点、错误信息、系统信息或用户信息。如果在检查点有不符合的情况被发现，你可以在测试结果窗口查看预期结果和实际测试结果。如果是位图不符合，你也可以查看用于显示预期值和实测结果之间差异的位图。如果由于测试中发现错误而造成测试运行失败，你可以直接从测试结果中查看有关错误信息。

### 9.3.4　负载压力自动化测试

**1．概述**

当一个企业自己组织力量或委托其他的软件公司开发了一套应用系统的时候，尤其

是以后在生产环境中实际使用起来时，用户往往会产生疑问，这套系统能不能承受大量的并发用户同时访问，随着生产数据的不断累积，系统的响应处理能力是不是会明显下降直至用户不能接受。这类问题最常见于采用联机事务处理（OLTP）方式的数据库应用、Web 应用和视频点播应用等系统。

比如，电信计费软件，众所周知，每月 20 日左右是市话交费的高峰期，全市几千个收费网点同时启动，收费过程一般分为两步，首先要根据用户提出的电话号码来查询出其当月产生的费用，然后收取现金并将此用户修改为已交费状态。一个用户看起来简单的两个步骤，当成百上千的终端同时执行这样的操作时情况就大不一样了，如此众多的交易同时发生，对应用程序本身、操作系统、中心数据库服务器、中间件服务器、网络设备的承受力都是一个严峻的考验。决策者不可能在发生问题后才考虑系统承受力。预见软件系统的并发承受力，这是在软件测试阶段就应该解决的。

如何模拟实际情况呢?找若干台电脑和同样数目的操作人员在同一时刻进行操作，然后拿秒表记录下反应时间?这样的手工作坊式的测试方法不切实际且无法捕捉程序内部的变化情况。这就需要负载压力测试工具的辅助。

那么，什么是负载测试和压力测试呢?负载测试是为了证明在与产品（预期）规模等同的数据库中处理给定的事务请求的容量下，系统功能与性能是否与需求规格说明中规定的，可接受的响应时间一致的测试过程。而压力测试则是使客户机在大容量情况下运行的测试过程，目的是查看应用将在何时何处出现中断，即识别系统的薄弱环节。压力测试中可能暴露的系统缺陷有内存泄露、系统资源过量消耗、磁盘空间用完等。

负载压力测试工具可以记录下客户端的操作，并以脚本的方式保存，然后建立多个虚拟用户，在一台或几台 PC 上模拟上百或上千虚拟用户同时操作的情景，同时记录下每一事务处理的响应时间、中间件服务器资源使用、数据库负载等，测试工程师可以根据测试结果分析系统瓶颈。

在各种类型的并发测试中，基于 Web 的应用占了很大的比例，现在有相当数量的联机事务处理类型系统采用 Web 方式，还有一些网站，对并发连接的数量和自己网站对大量用户访问的支持能力，都表示出了相当程度的关心。对于负载压力测试工具，它提供了对 Web 页面压力测试的完整解决方案，包括用户模拟、Web 服务器监控、页面每秒钟点击率统计、单独页面加载时间分析等各种专门针对 Web 的特性。随着服务器端处理任务的日益复杂以及网站访问量的迅速增长，服务器性能的优化也成了非常迫切的问题。在优化之前，最好能够测试一下不同负载条件下服务器的性能表现。定位性能瓶颈所在，是设计性能改善方案之前的一个至关紧要的步骤。

负载测试是任何 Web 应用的开发周期中一个重要的步骤。如果你在构造一个为大量用户服务的应用，搞清楚你的产品配置能够承受多大的负载非常重要。如果你在构造一个小型的 Intranet 网站，测试能够暴露出最终会导致服务器崩溃的内存漏洞、资源竞争等情况。但是在实际的开发过程中，要按照实际投入运行的情况，组织成千上万的用户

来进行压力测试，无论从哪个方面看，都是不现实的。而且一旦发现了问题，不仅需要重复地进行这种耗费巨大的测试，而且问题不容易重现，不能方便地找出性能的瓶颈所在。而使用测试工具进行压力测试就不会存在这种情况。

无论是哪种情形，花些时间对应用进行负载压力测试可以获得重要的基准性能数据，为未来的代码优化、硬件配置以及系统软硬件升级带来方便。即使经费有限的开发组织也可以对它们的网站进行负载压力测试，因为有些测试工具是可以免费下载的。

在测试阶段使用负载压力测试工具进行测试，还可以模拟数据库死锁情况，结合压力分析 SQL 效率，优化应用程序和数据库配置等工作，使软件系统更加健壮和高效。这样的实例也很多，比如有公司在测试某省的大型电信业务网上受理系统时，200 个并发用户同时联机时速度正常，但当达到用户量达到 500 个的时候，受理速度明显变慢，通过监控发现 Web 服务器的流量有所降低，而表空间对应的数据文件中发生的磁盘物理读的次数却大于正常水平，最后通过诊断确定有部分复杂的 SQL 查询（如大表连接操作、嵌套查询等）没有利用合适的索引和采用最优的解释方案，而造成全表扫描。而且数据库配置参数 DB-BLOCK-BUFFERS 太小，不能适应 500 个用户或更大规模的并发情况。经过测试人员和开发人员对系统的共同调整，再次测试的时候一切恢复正常，500 个用户的并发测试顺利通过。

**2．测试原理**

负载压力测试工具基本上都是采取录制回放的方式来模拟用户的实际操作的，而且测试工具一般都会有一个后台代理进程，通过该代理进程，测试工具可以监视并获取在各种通信协议下应用系统的客户与服务器端的通信信息，测试工具会用一种类 C 或者其他的脚本语言（TSL）生成一个测试脚本，该脚本记录了你对服务器的请求过程，然后测试工具就可以回放刚才的访问过程，接收服务器的响应，当然你也可以用手工编程生成这个脚本。

通常情况下，实施负载压力测试的工作示意图如图 9-1 所示。

当被测系统运行时，脚本生成器会自动获取系统客户端与服务器的通信信息并转换成测试工具可以识别的脚本，测试工具通过控制台将测试脚本分发到其他负载生成器上以模拟多用户对服务器的并发访问，同时，控制台还可以通过服务器上开启的远程 RPC 服务，获取相关的资源使用信息，最后可以收集测试数据。

在进行测试脚本录制与分配的过程中，应遵循如下几个原则。

（1）脚本越小越好。就像编写程序代码一样，不要太长，尽量做到一个功能一个脚本。如果那些功能是连续有序的，必须先做上一个，才能执行下一个，那就只好放在一起了。

（2）选择负载压力最高的业务功能进行测试。有些人总希望能够测试几乎所有的功能，但事实上，这种想法不可行，因为测试是需要投入时间和金钱的，我们应该结合用户实际的使用情况，选择负载压力最高的业务功能进行测试，以满足性能测试的需求，

达到测试的预期目的。

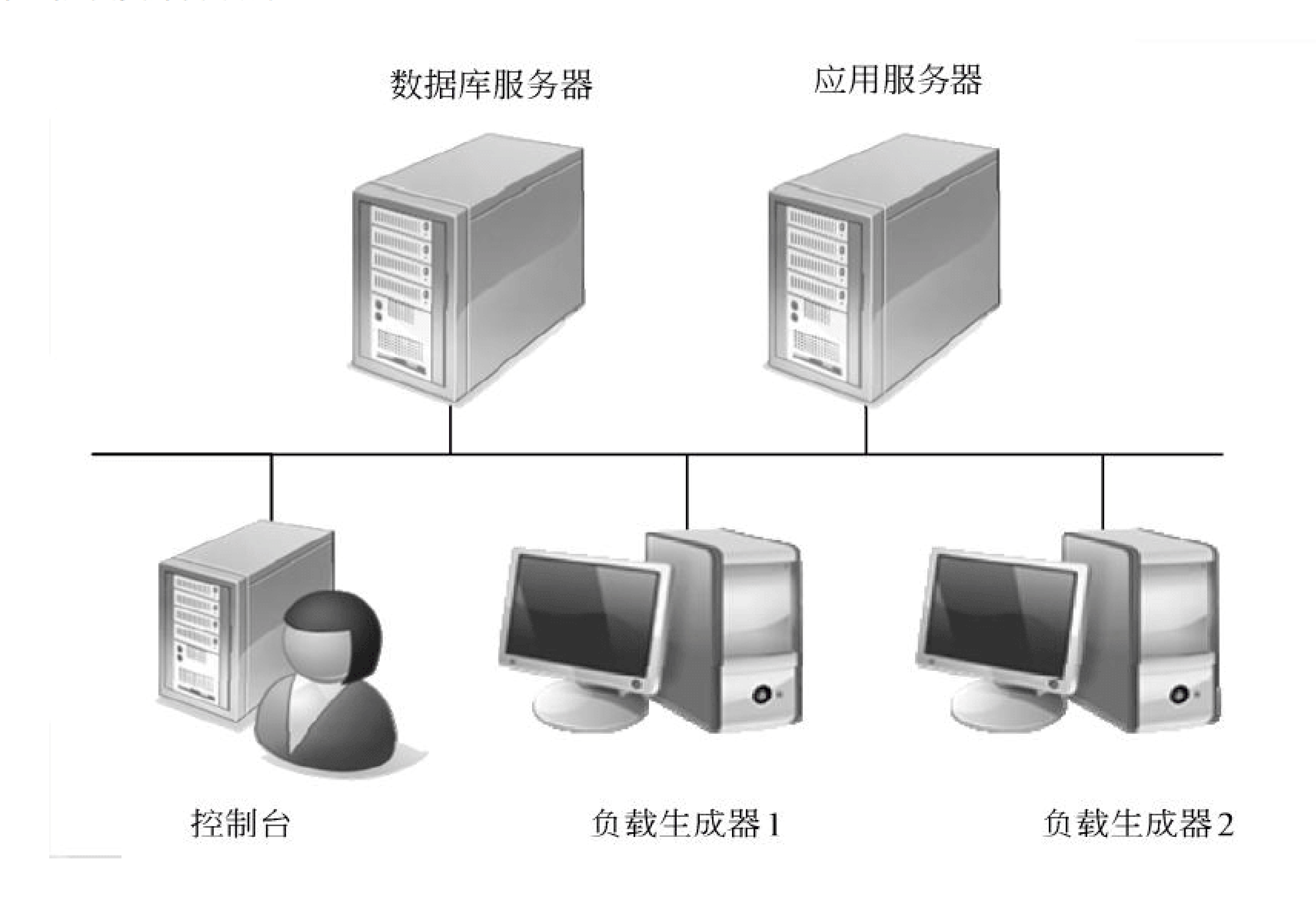



图 9-1 实施负载压力测试的工作示意图

（3）选择所需要的操作进行录制。例如，需要测试服务器承受负载压力的性能，某些客户端的操作不会对后台服务器产生负载压力，就可以不录制。比如一些查询操作，选择查询条件的页面可以不录制，但对于一些页面有可能要与后台服务器传递参数，就需要录制了。如何确定哪些操作需要录制，一是可以找开发人员了解清楚程序设计结构和运行模式，二是可以靠自己的经验，熟能生巧当然不会错。

由于负载压力测试工具是一种预测系统行为和性能的负载测试工具。它需要模拟成百上千甚至上万的用户同时操作应用系统，实施并发负载及实时性能监测。通常情况下，测试工具模拟多用户并发访问可以有以下两种方式。

（1）进程回放模式：模拟多进程运行方式，即客户端与服务器的访问采用进程方式，每一个虚拟用户通过一个进程建立与服务器的通信连接并访问。

（2）线程回放模式：模拟多线程运行方式，即客户端与服务器的访问采用线程方式，每一个虚拟用户通过一个线程建立与服务器的通信连接并访问。

不论测试工具采用的是哪种录制回放模式，其必须经历的几个操作步骤如下。

（1）协议选择：如前所述，测试工具都是通过获取在各种通信协议下应用系统的客户端与服务器端的通信信息并进一步来实现负载压力测试的，所以首先要确定被测应用系统客户端与服务器端的通信所使用的协议类型。一般而言，B/S 系统选择 Web（Http/Html），两层 C/S 系统则根据 C/S 结构所用到的后台数据库来选择不同的协议，协议的选择请参考有关章节。

（2）创建测试脚本：选择好相应的录制协议以后，就可以启动脚本录制工具，通过测试工具的代理进程获取应用系统客户端和服务器端的通信信息，测试工具可以自动记录客户端对服务器端的访问操作并自动转换成所需的脚本代码，还可以直接编辑测试脚本以满足各种复杂测试的需求。

（3）参数化测试数据：对于创建的测试脚本，可以利用数据池技术对其中的某些数据参数化，从而更好地模拟真实应用访问。以一个订单输入过程为例，参数化操作可将记录中的常量数据，如订单号和客户名称，由变值来代替，以更好地模拟多个实际用户的操作行为。

（4）创建虚拟用户，设定负载方案：由于负载压力测试的目的就是要模拟多用户并发访问应用系统，所以在开始测试之前就应该设计好需要模拟的虚拟用户数，然后使用测试工具控制台设定负载方案。

（5）执行测试：设定了相应的负载测试方案后，就可以开始测试了，测试过程中，测试工具会自动记录测试结果，包括事务处理的响应时间，服务器的资源占用情况等。

（6）结果分析：测试结束后，测试工具会收集汇总所有的测试数据并生成测试结果、报告，这些数据主要包括交易性能数据（如响应时间等）、服务器资源占用情况、网络设备和数据库的实时性能数据等。通过这些数据就可以从客户端、服务器端以及网络三方面来评估系统组件的运行性能，从而定位应用系统存在的主要问题，为系统改进做准备。

# 第 10 章　搭建缺陷管理系统

**学习目标：**

1．了解 Bug 管理流程及工具介绍

2．掌握 Bugzilla 的安装配置

3．熟悉缺陷报告的提交过程

## 10.1　Bug 管理流程及工具介绍

在软件测试工作过程中，始终围绕软件的 Bug 管理流程这个主题。软件测试人员的职责是找出软件系统中存在的各种缺陷，以及跟踪处理这些缺陷。所以，从测试管理角度来讲，如何有效地管理发现的 Bug 将直接影响软件的质量与工作效率。

前面章节已经介绍过 Bug，这里就不再重复。大多数公司采用下面的 Bug 处理方式，这样的方式从测试管理角度来讲，起到了层层监控、步步把关的作用。下面详细分析一下该流程。

Bug 管理流程是以参与人员的角色分工的，这里主要从测试人员、测试组长、开发组长、开发人员、项目经理等角色考虑。

**1．测试人员**

测试人员发现 Bug 后，利用 Bug 管理工具添加 Bug，此时 Bug 的状态为 New（新建），在添加 Bug 时需指定该 Bug 的下步处理人，一般情况下为当前项目的测试组长。

**2．测试组长**

测试组长查看对应项目中需要自己处理的 Bug，进行 Bug 的 Review（审查）工作，检查测试组员新增的 Bug 是否符合规范，比如语言描述是否清晰；问题定位是否准确等，或者判断该问题是否确实是一个 Bug，还是因组员不熟悉需求、理解偏差而引起的误提。如有问题，则将该 Bug 返回给 Bug 提交者，让其修改后再提交给测试组长。如无问题，则将该 Bug 提交给开发组长。

**3．开发组长**

开发组长处理指派给其的 Bug。根据 Bug 的所属功能模块标识，分派给相应的开发人员，如果在这个过程中，开发组长认为某些 Bug 提交的有问题，则可返回给测试组长，并加上相应的说明（Comment），由测试组长再次审查。测试组长则会与测试组员共同处理该 Bug。如无问题，则可再次指派给开发组长。

**4．开发人员**

开发人员处理开发组长指派给其的 Bug。根据 Bug 的描述重现并修复 Bug，修复后将该 Bug 状态置为 FIXED（已修复），并指派给相应的 Bug 提交者，由 Bug 提交者在后续版本中进行验证。如果开发人员认为该问题不是一个 Bug，则可向开发组长反映，或者咨询 Bug 的提交者；测试人员将开发人员 Fix 回来的 Bug 进行验证，如果该 Bug 被成功修复，则关闭该 Bug，如果经检查未能成功修复，则 REOPENED （重新打开）该 Bug，继续按照 Bug 的处理流程流转。

**5．项目经理**

当对提交的 Bug 有分歧时，可由项目经理、测试组长、开发组长等进行 Bug 的评审，并商定问题如何处理，是否作为保留或是当前版本不改，需给出一个定论。Bug 的处理流程如图 10-1 所示。

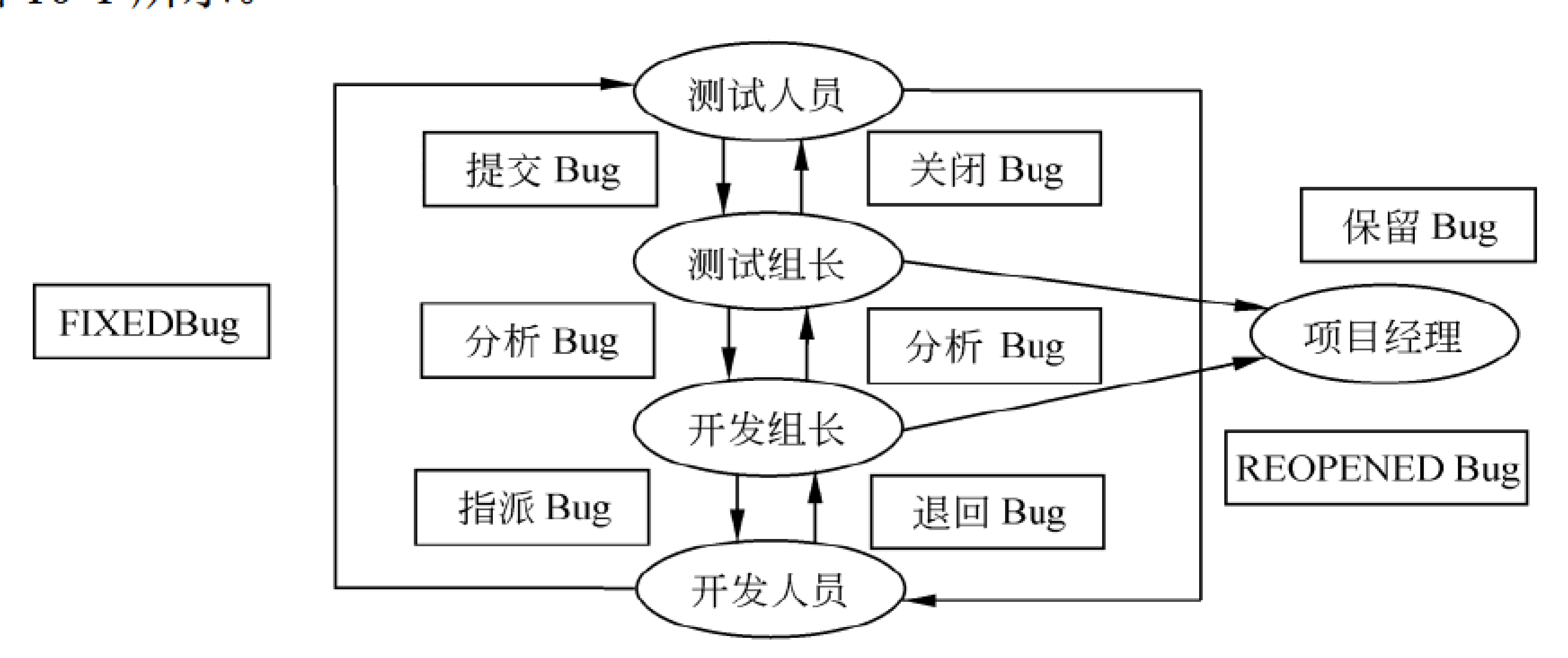



图 10-1　Bug 处理流程

一般来讲，Bug 的处理是一个循环反复的过程。当出现争议的时候，必须由项目负责人参与 Bug 的处理，而不能由开发组或者测试组单方面决定 Bug 的终止。

| |
|---|
| 注意：任何人都不应具备删除 Bug 的权限。 |

有了清晰的 Bug 管理流程，如果再辅以先进完善的 Bug 管理工具，软件测试的工作就能顺利地开展，并能在一定程度上提高我们的工作效率。那么，一般有哪些 Bug 管理的工具呢?

目前主要的缺陷管理工具有 TestDirector、Bugzilla、TestTrack、Mantis 等，另外有些公司自己开发了一些管理工具。其实不管采用什么工具，只要能高效地协同软件测试工作就行。本章主要讲解一种免费的 Bug 的管理工具 Bugzilla。

## 10.2　Bugzilla 工具的安装配置

Bugzilla 是一个缺陷跟踪系统，用于对软件产品程序开发过程的缺陷跟踪。它的强

大功能表现在以下几个方面：

- 强大的检索功能。
- 用户可配置的通过 E-mail 公布 Bug 变更。
- 历史变更记录。
- 通过跟踪和描述处理 Bug。
- 附件管理。
- 完备的产品分类方案和细致的安全策略。
- 安全的审核机制。
- 强大的后端数据库支持。
- Web、Xml、E-mail 和控制界面。
- 友好的网络用户界面。
- 丰富多样的配置设定。
- 版本间向下兼容。

Bugzilla 是一个拥有强大功能的缺陷跟踪系统，它可以使我们更好地在软件开发过程中跟踪处理软件的缺陷，为开发和测试工作以及产品质量的度量提供数据支持，从而有效地保证软件产品的质量。针对很多用户无法成功安装配置 Bugzilla 的现象，本章重点讲解如何在 Windows Server 2003 上安装配置 Bugzilla 工具及搭建一个简单 SMTP 服务器。Bugzilla 的使用方法比较简单，大家可以从网络上找到相关的资料自行学习，亦可参考其说明文档，学习 Bugzilla 的使用方法。

### 10.2.1 Bugzilla 的安装配置

Bugzilla 一般在 Linux 平台上安装使用，当然也可在 Windows 平台上安装，它的安装过程较为复杂，需要 ActivePerl、MySQL 软件支持，以 IIS 或者 Apache 作为 Web 服务器，如果公司无邮件服务器，还需配置一个局域网内的邮件服务器。下面介绍 Bugzilla 在 Windows Server 2003 上的安装配置。在安装开始之前，建议设置密码时不要将密码设置得太复杂，避免遗忘密码，而且也没有必要。

本次安装所需软件版本分别为：

ActivePerl：ActivePerl-5.8.8.822-MSWin32-x86-280952.msi
MySQL：MySQL-5.0.18.exe
Bugzilla：Bugzilla-3.1.3.tar.gz
Bugzilla 汉化包：Bugzilla-3.1.2-cn.zip

**1．安装 ActivePerl**

安装步骤

（1）双击 ActivePerl.5.8.8.822-MSWin32-x86-280952.msi，单击图 10-2 中的 Next 按钮。

（2）在图 10-3 所示的窗口中选择 I Accept the Terms in the License Agreement 单选按

钮，单击 Next 按钮。

（3）在图 10-4 所示的窗口中默认设置，单击 Next 按钮。




图 10-2　ActivePerl 准备安装界面

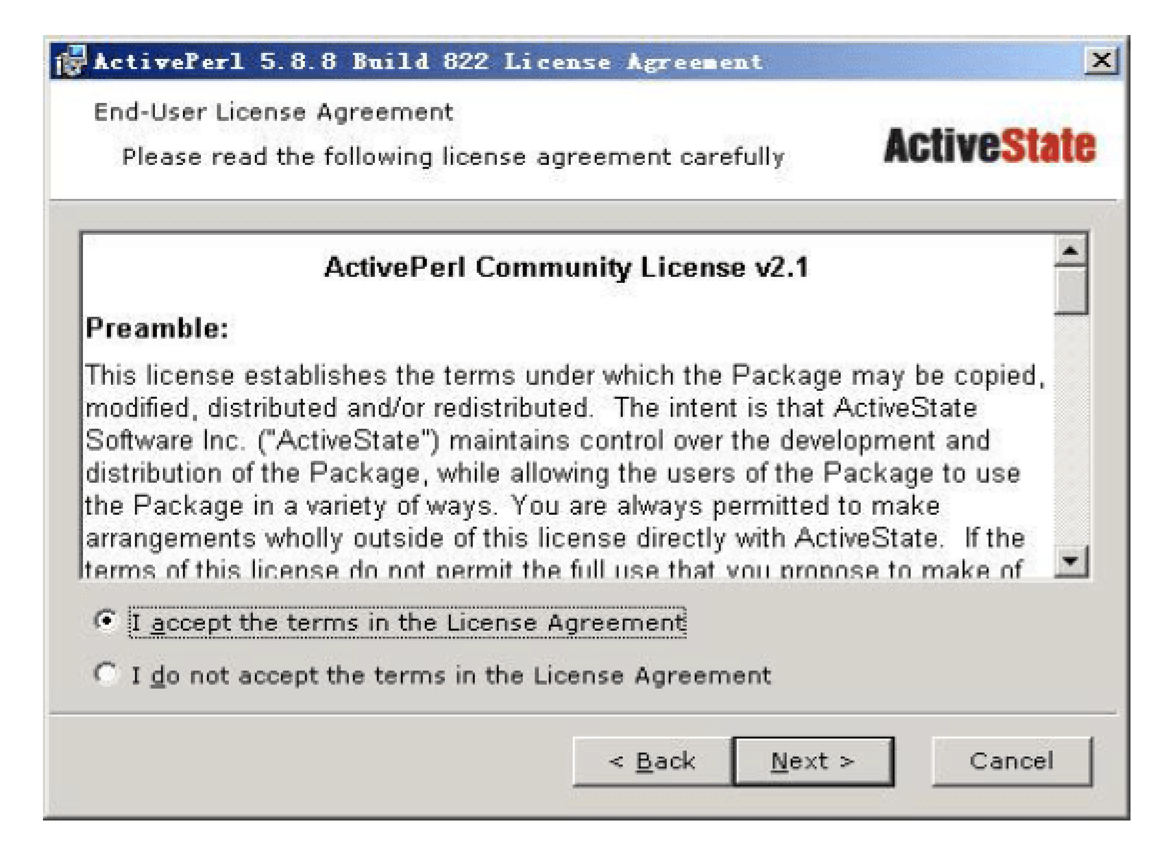



图 10-3　AcfivePerl 许可证界面




图 10-4　设置安装目录界面

（4）在图 10-5 所示的窗口中默认设置，单击 Next 按钮。

（5）在图 10-6 所示的准备安装界面中单击 Install 按钮，开始安装。

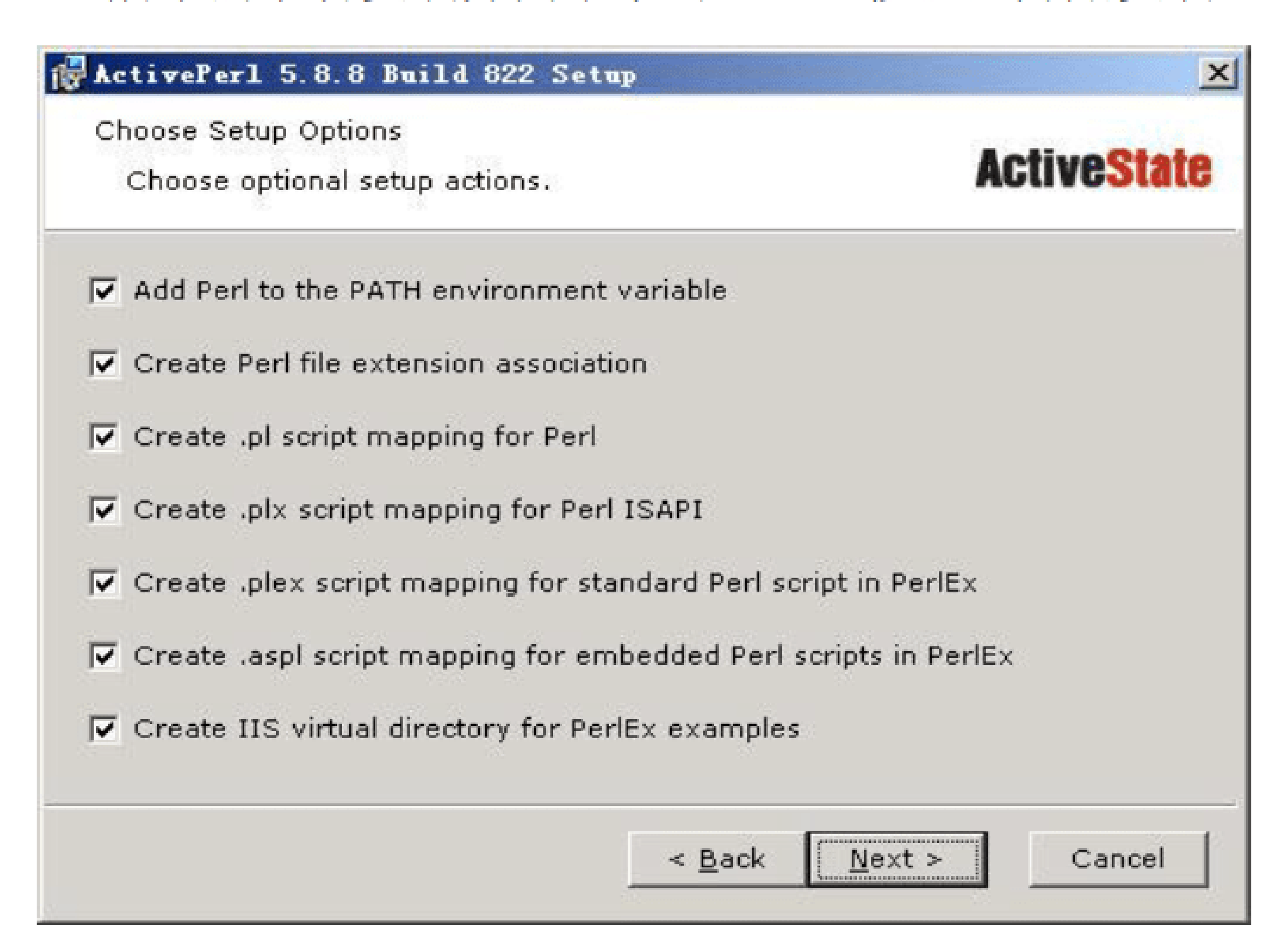



图 10-5　安装选项设置

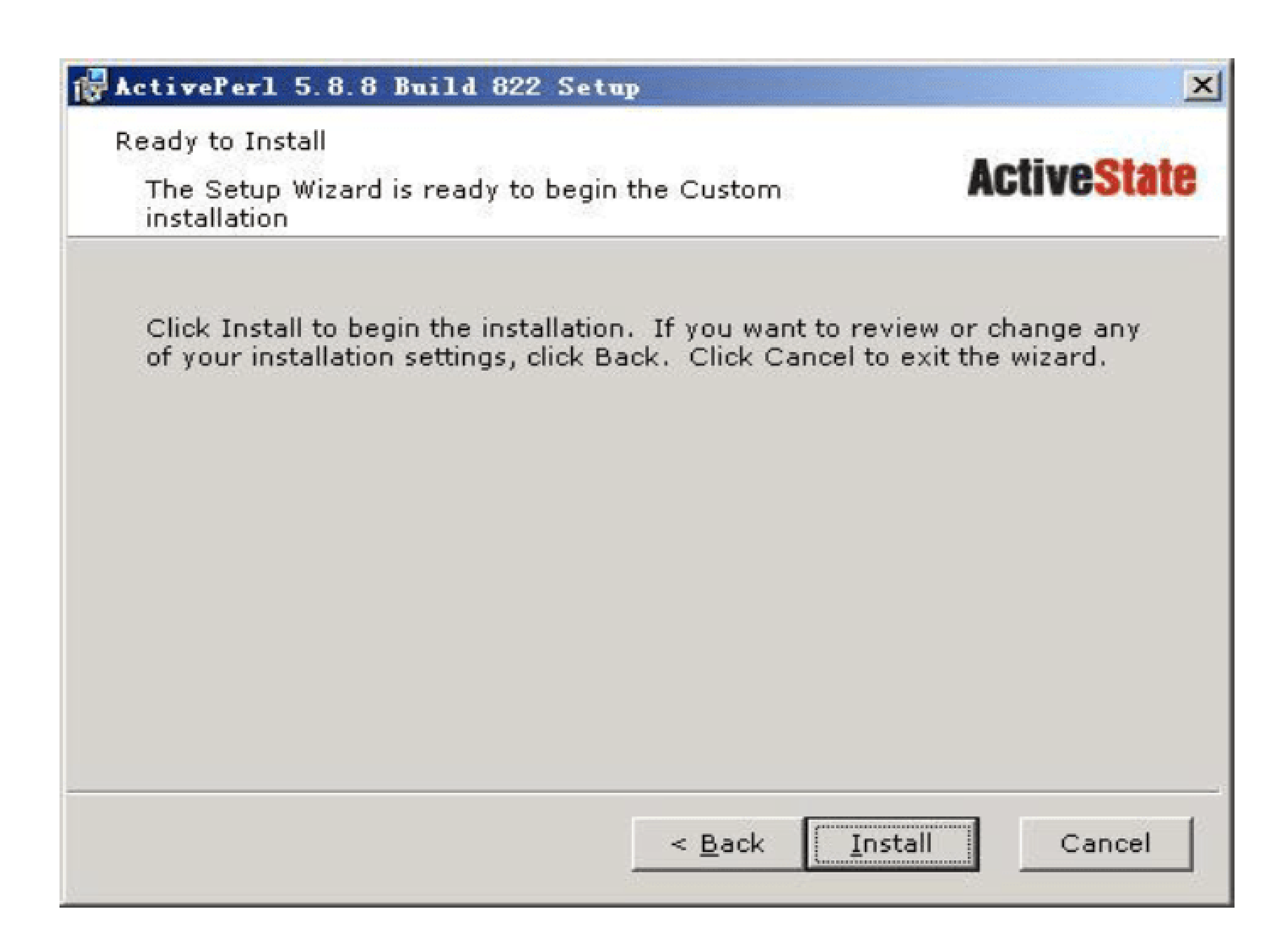



图 10-6 准备安装界面

（6）安装进程如图 10-7 所示。

（7）在图 10-8 所示的窗口中取消 Display the release notes 的勾选，单击 Finish 按钮，完成 ActivePerl 的安装。

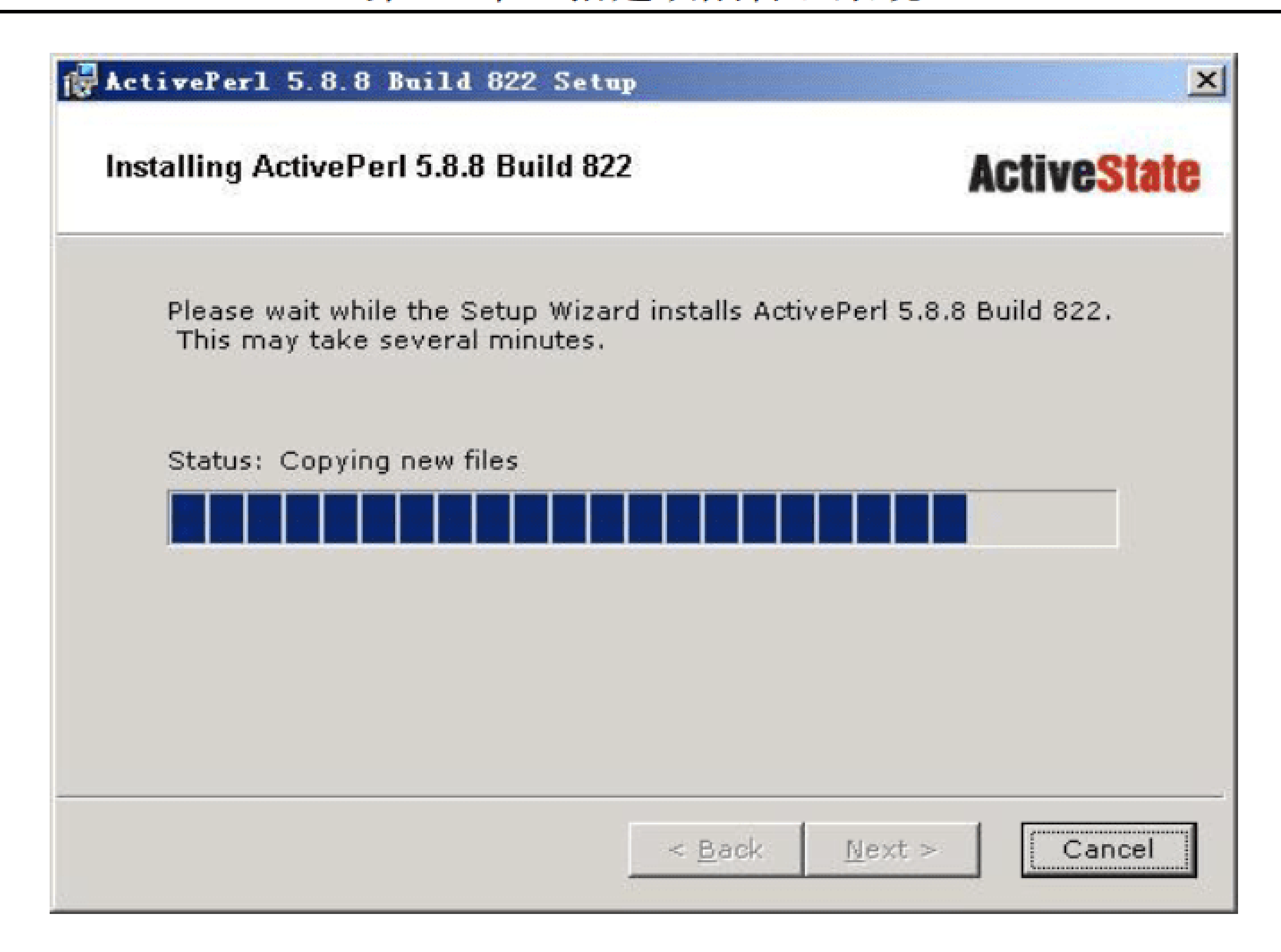



图 10-7　安装进程

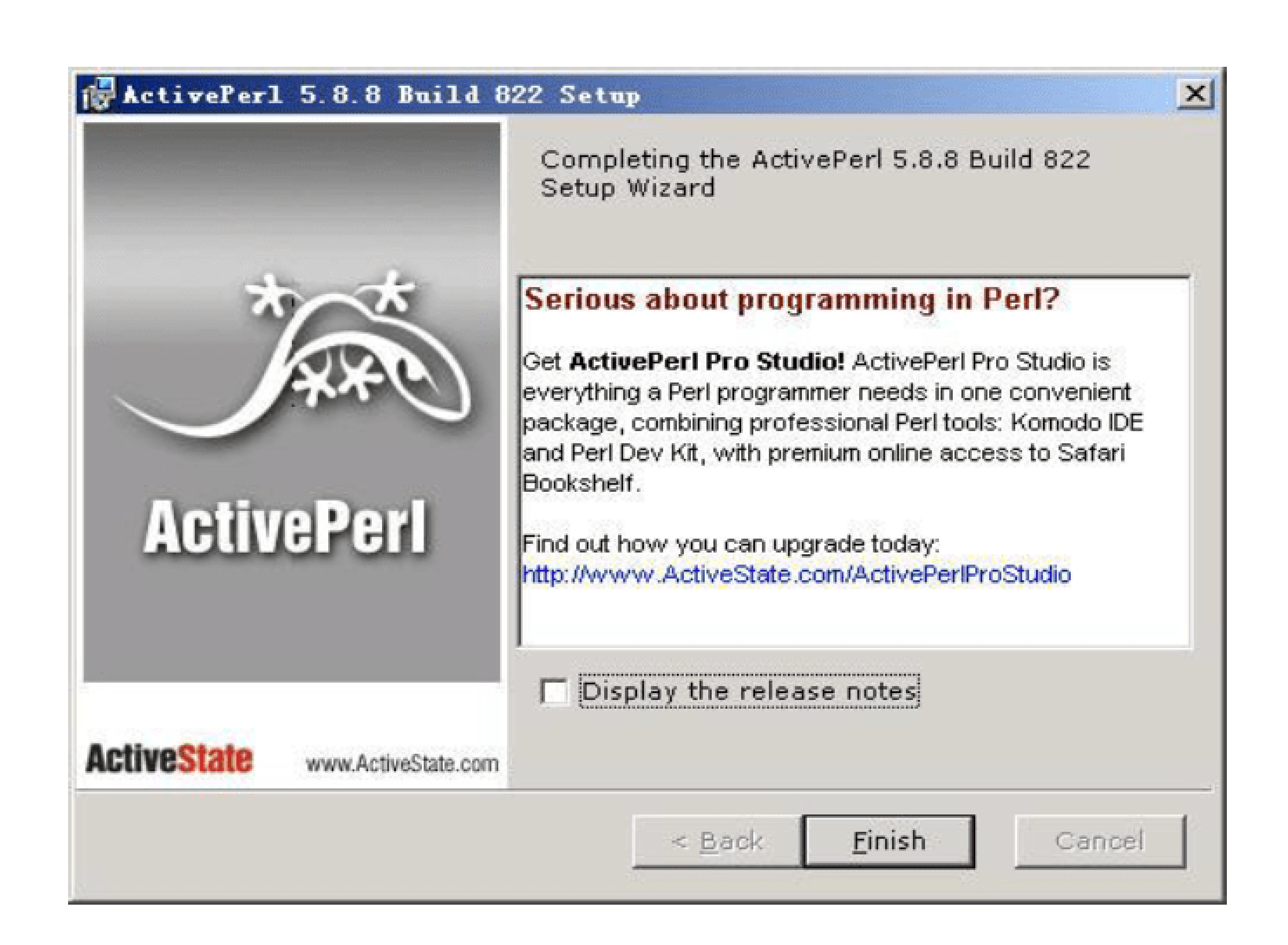



图 10-8　完成安装

## 2．安装配置 MySQL

安装步骤

（1）双击 MySQL-5.0.18.exe，单击图 10-9 所示窗口中的 Next 按钮。

（2）在图 10-10 所示的安装类型窗口中选择 Custom 单选按钮，单击 Next 按钮。

（3）出现图 10-11 所示的窗口，单击 Change 按钮，修改安装路径。




图 10-9 MySQL 准备安装

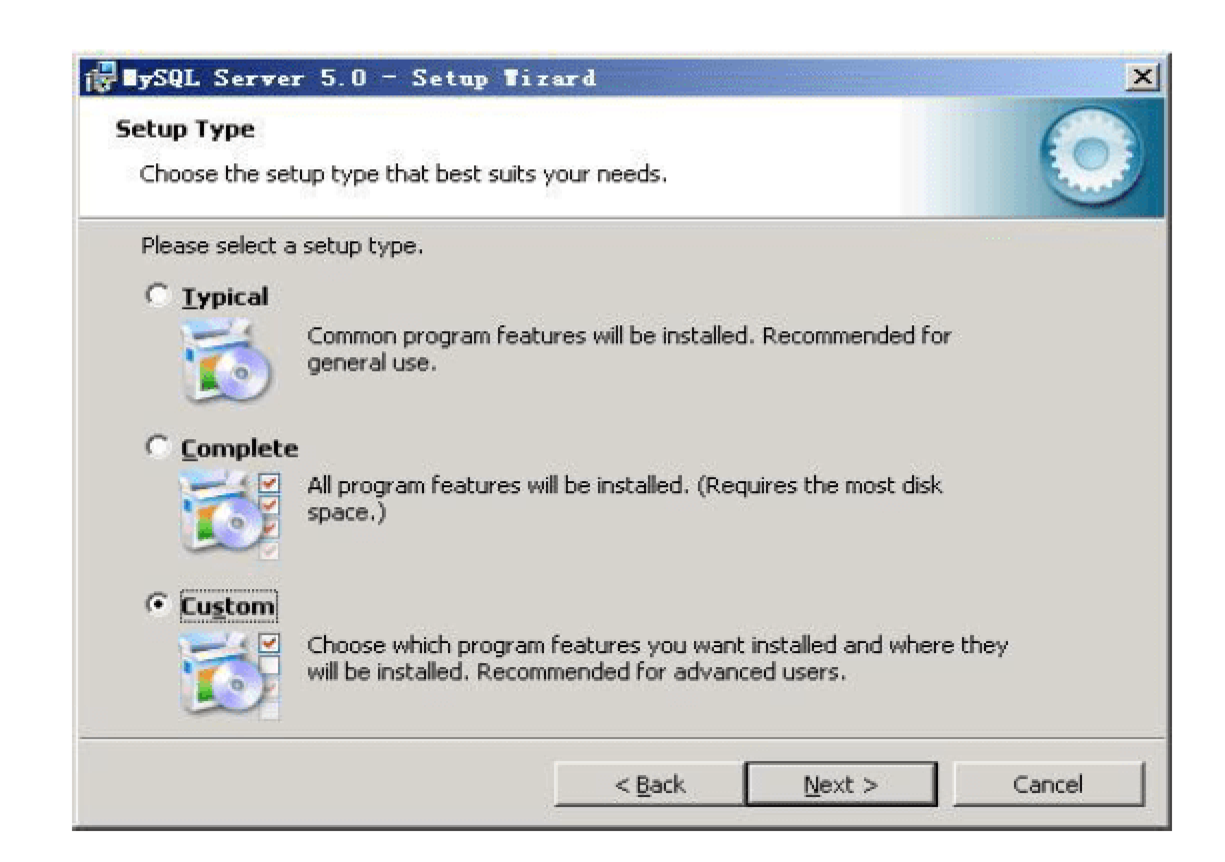



图 10-10 安装类型选择

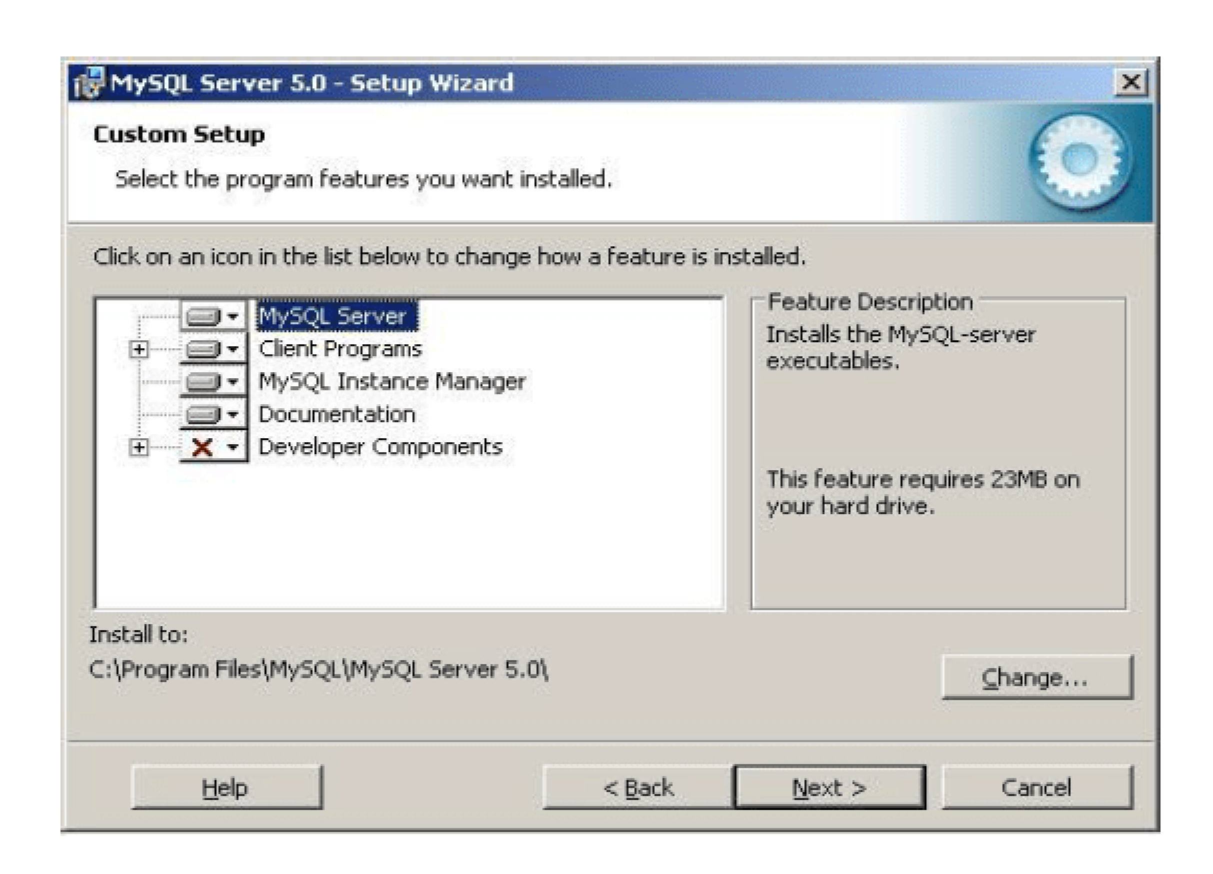



图 10-11 安装路径设置

（4）按图 10-12 所示，修改安装路径，建议修改为某系统盘的根目录，便于管理，如此处设置为 C:\mysql，设置完成后单击 OK 按钮。

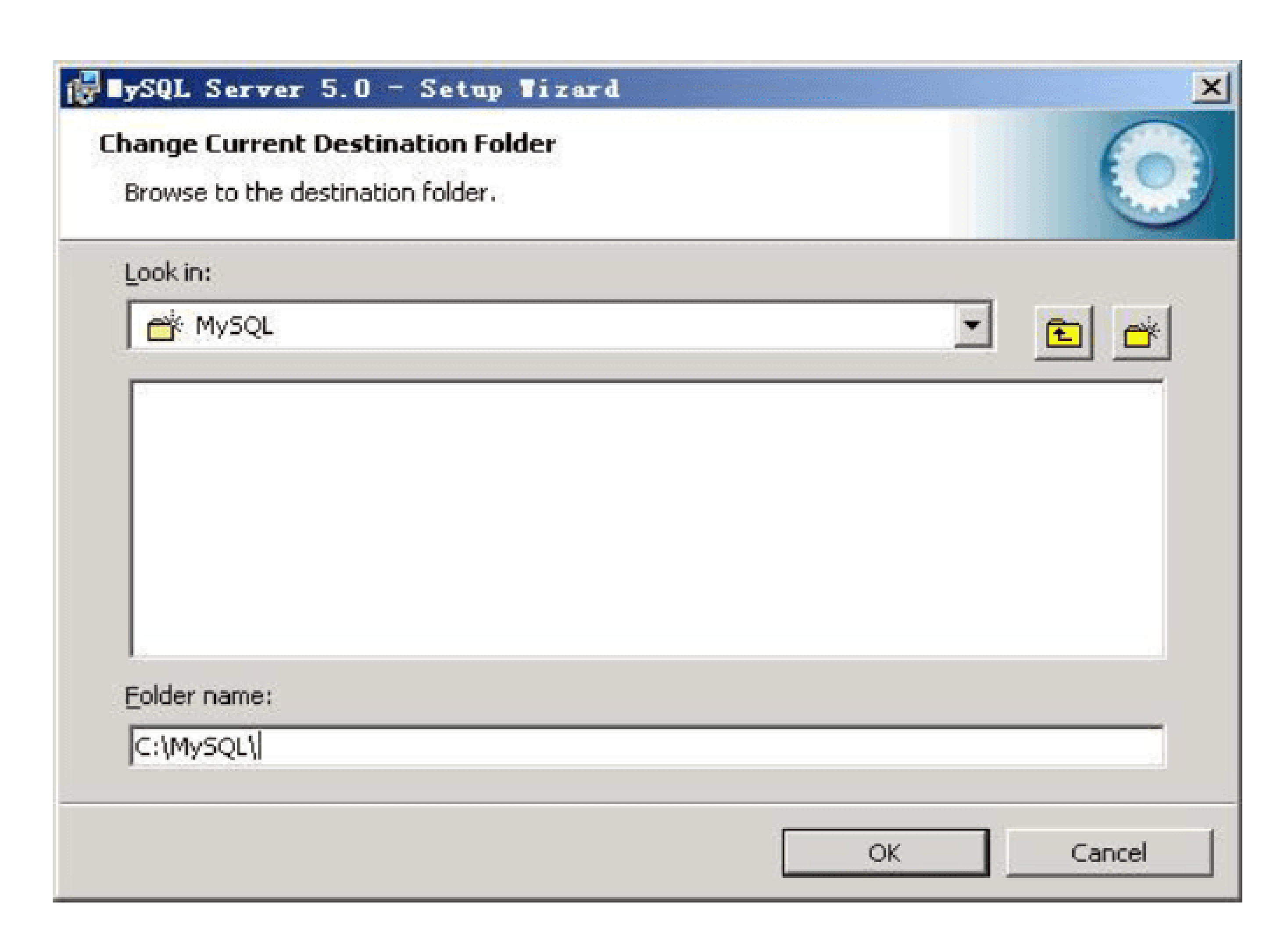



图 10-12　修改安装路径

（5）修改安装路径后，单击 Next 按钮，如图 10-13 所示。

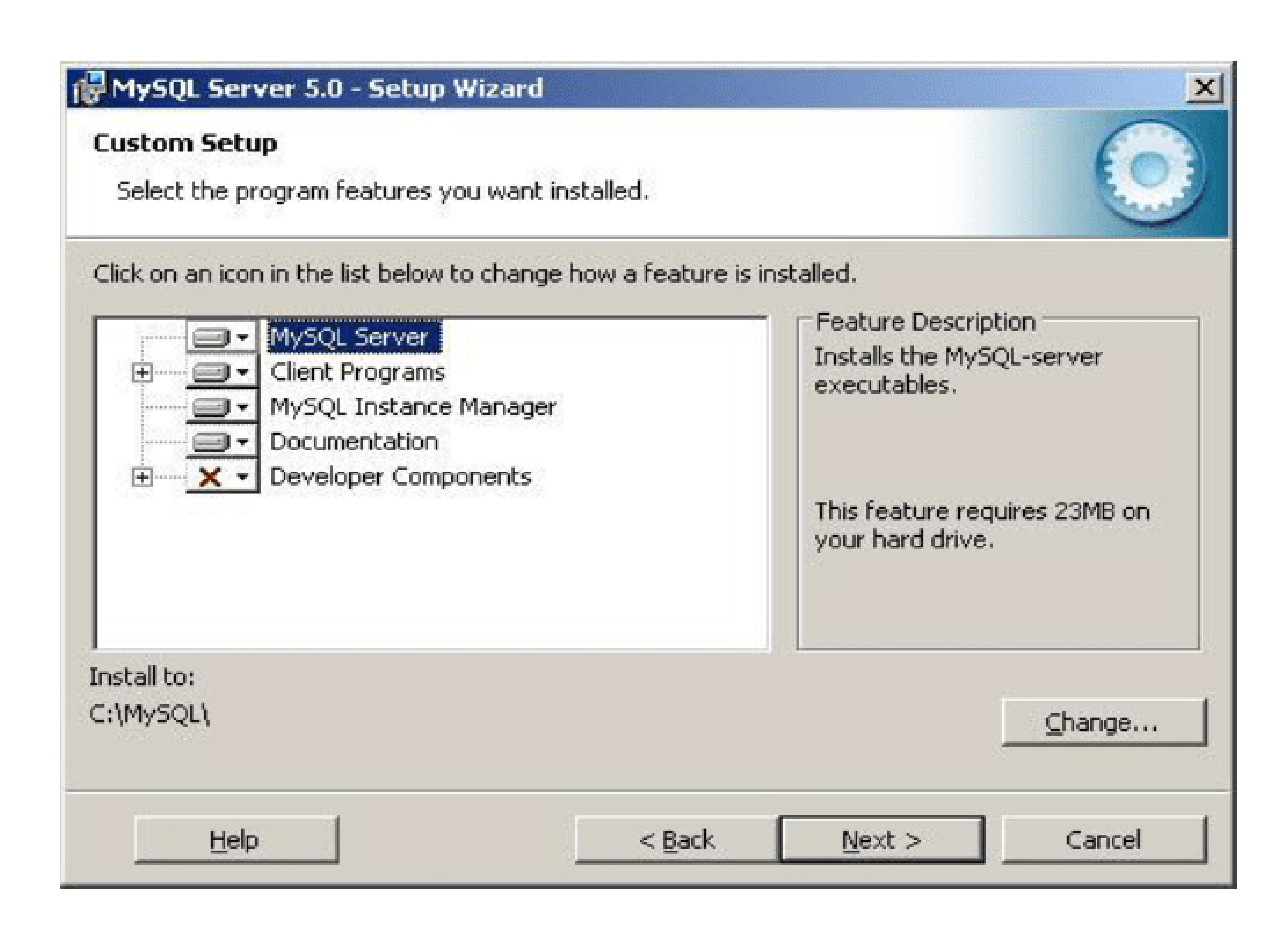



图 10-13　继续安装

（6）单击 Install 按钮，执行安装，如图 10-14 所示。

（7）安装进程如图 10-15 所示。

（8）选择 Skip Sign-Up 单选按钮，跳过注册，单击 Next 按钮，如图 10-16 所示。

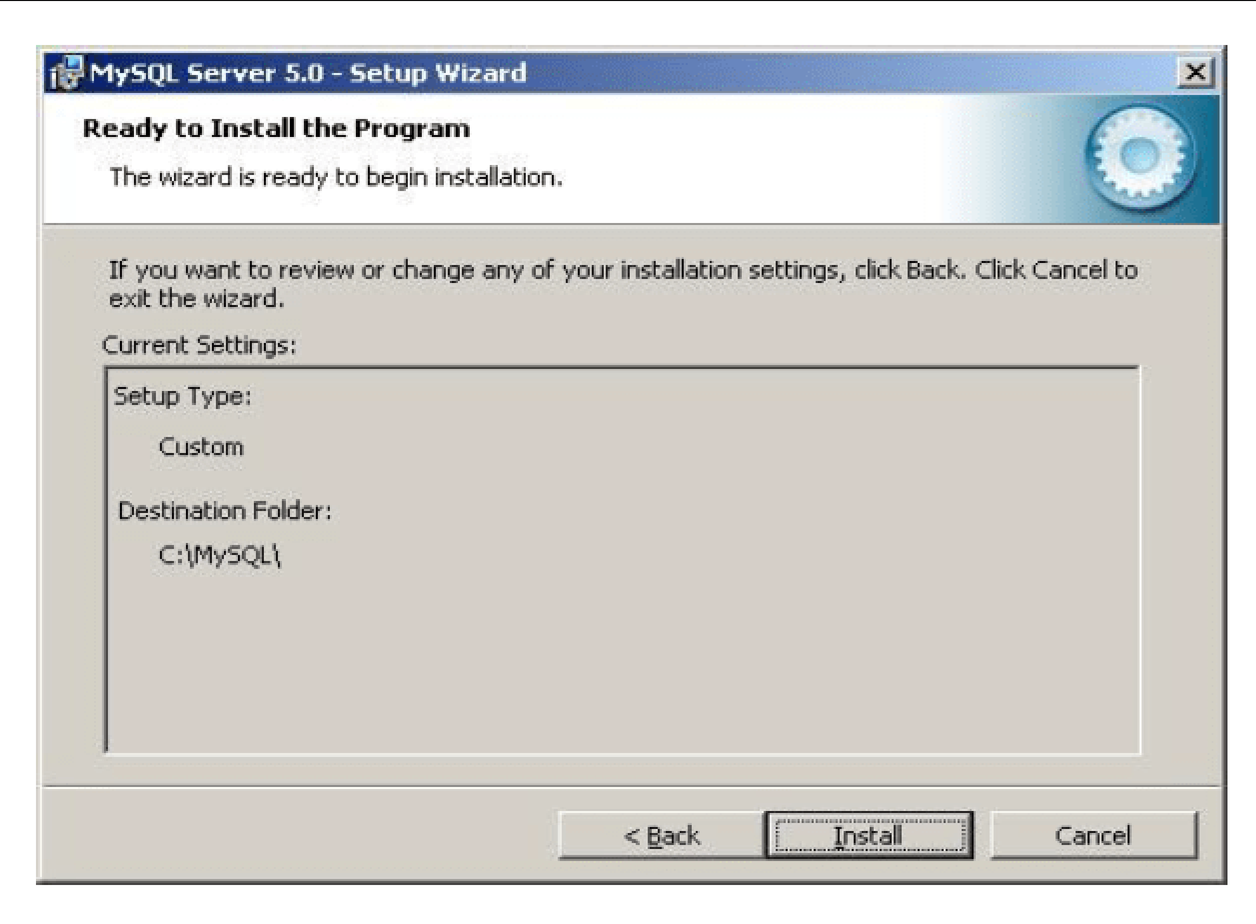



图 10-14　开始安装

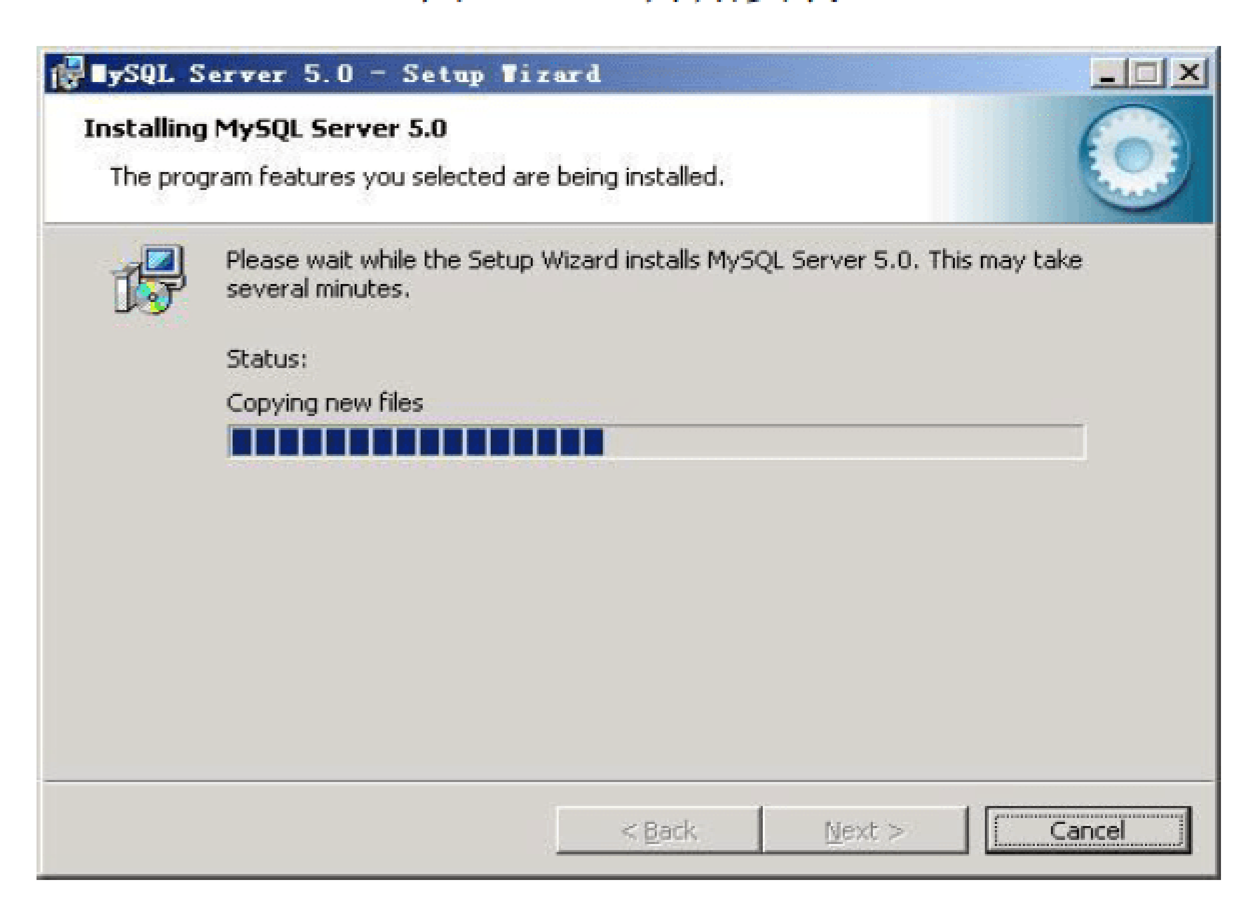



图 10-15　安装进程

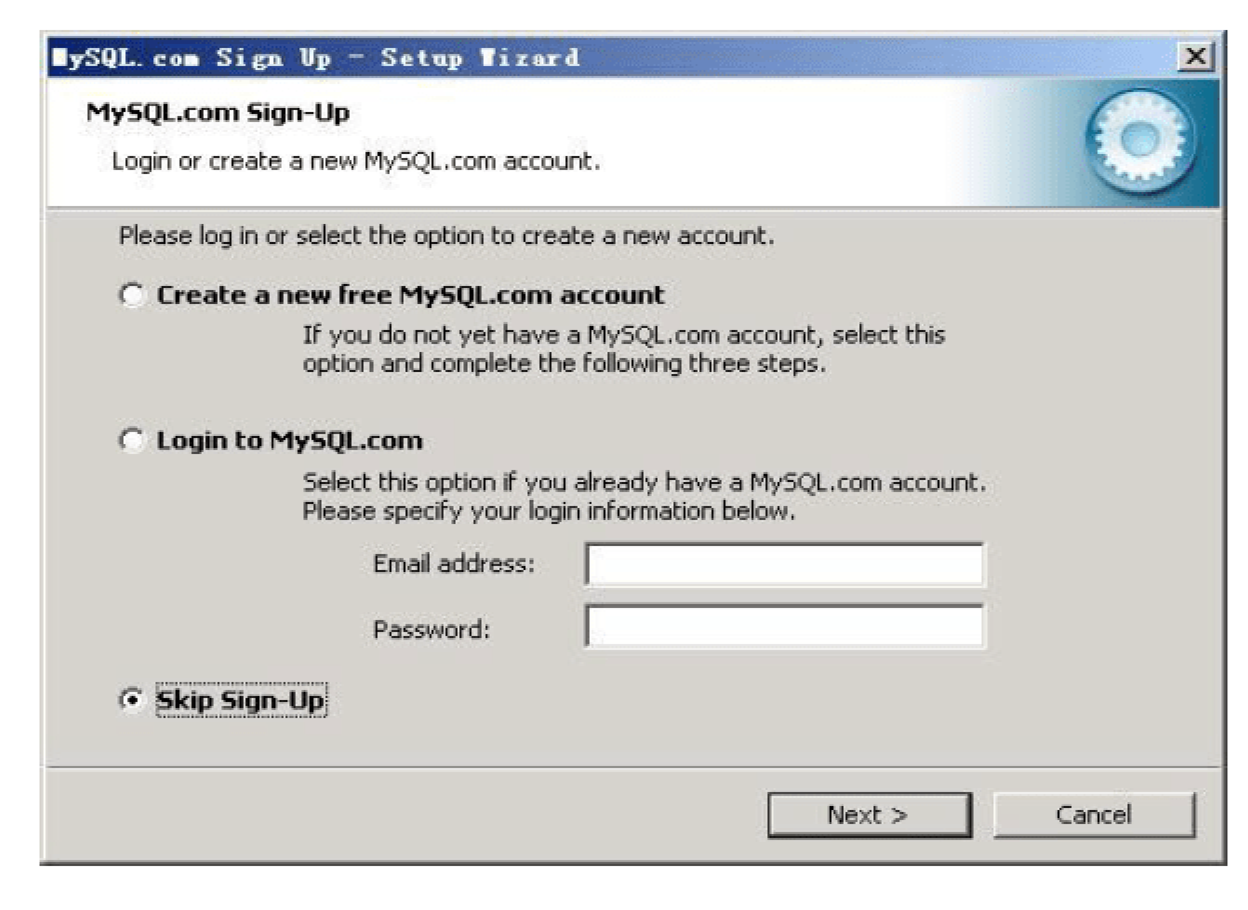



图 10-16　MySQL 注册界面

（9）勾选 Configure the MySQL Server now，单击 Finish 按钮，如图 10-17 所示。




图 10-17　完成安装

（10）进入 MySQL 配置欢迎界面，如图 10-18 所示，单击 Next 按钮。




图 10-18　进入 MySQL 配置界面

（11）按图 10-19 所示，选择 Standard Configuration（标准配置）单选按钮，单击 Next 按钮。

（12）按图 10-20 所示进行设置，单击 Next 按钮。

（13）在图 10-21 中勾选所有项，并设置 root 用户密码，单击 Next 按钮。

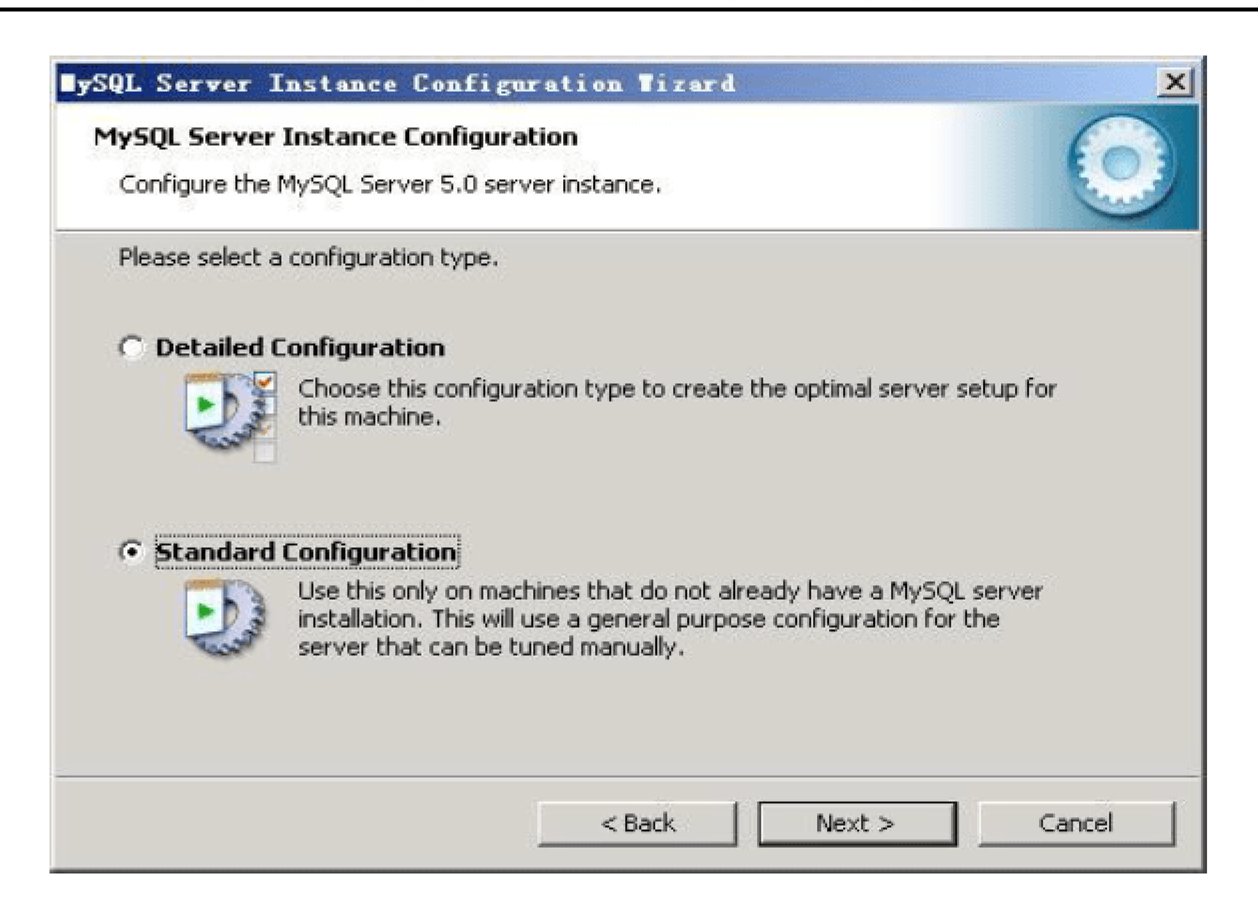



图 10-19 选择配置类型




图 10-20 配置 MySQL 界面




图 10-21 设置 MySQL 账号信息

（14）在图 10-22 所示窗口中单击 Execute 按钮，执行配置。

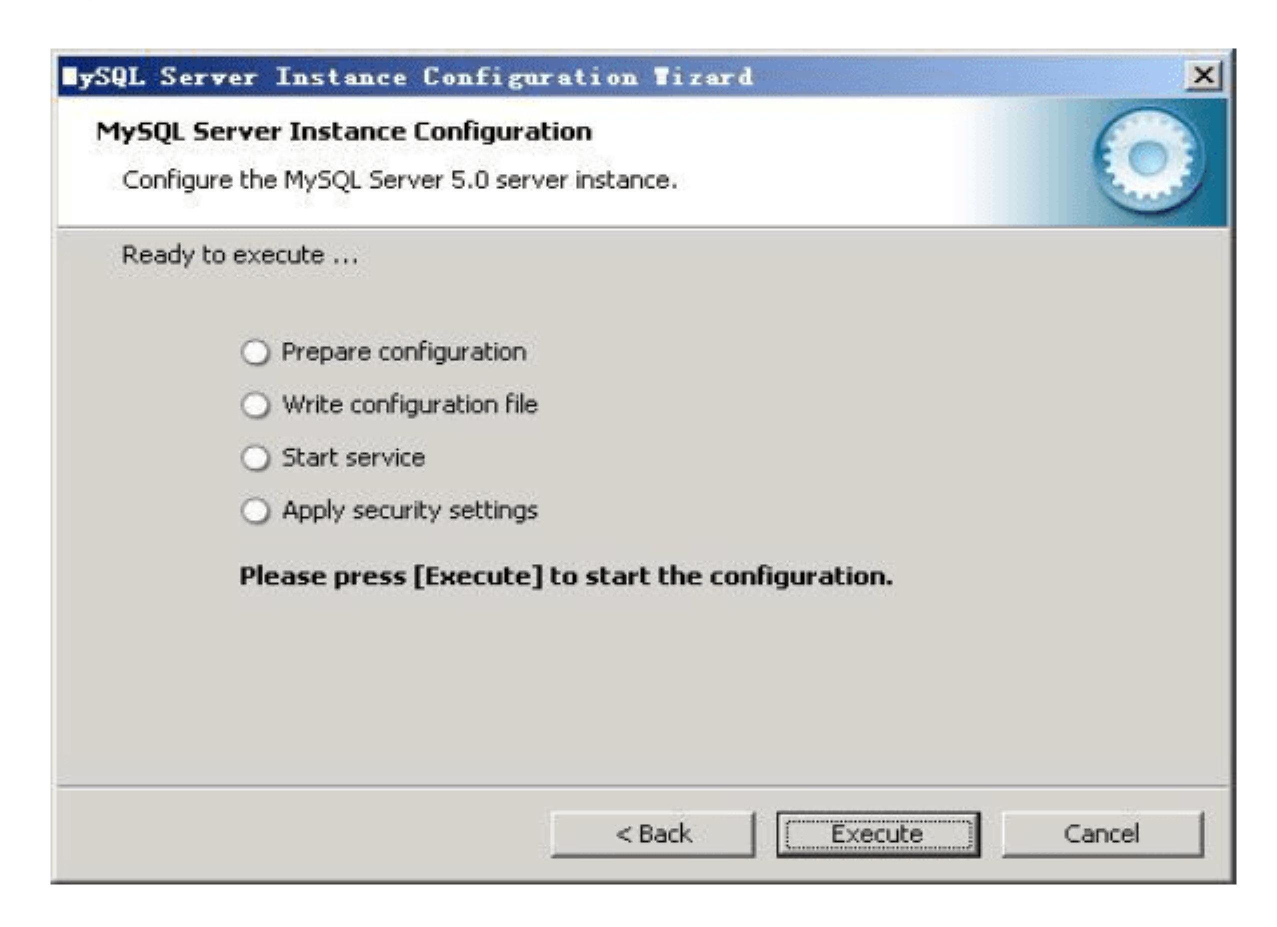



图 10-22　MySQL 配置界面

（15）单击 Finish 按钮，完成 MySQL 的安装与配置，如图 10-23 所示。

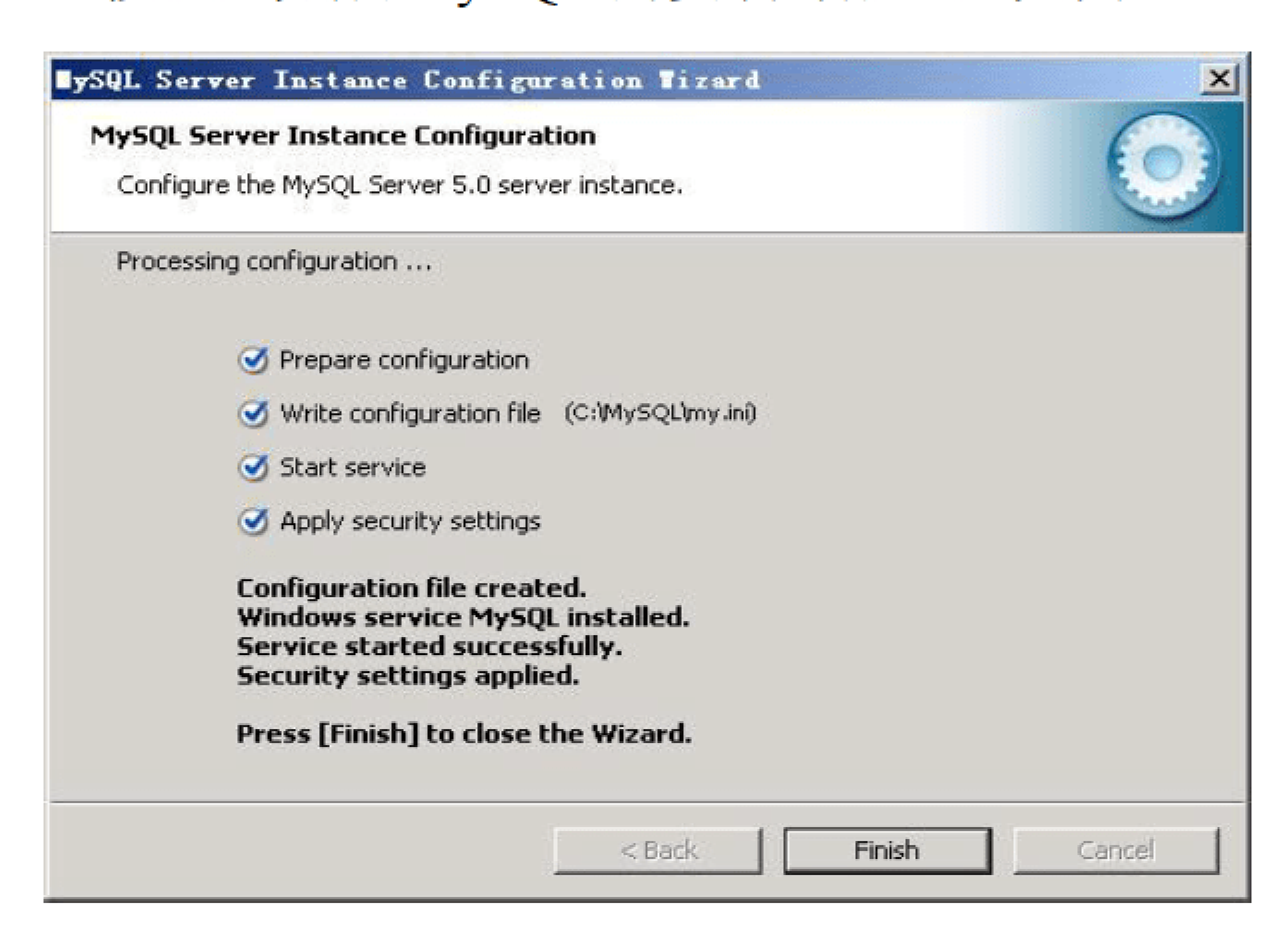



图 10-23　MySQL 安装完成

（16）打开命令提示符，进入 MySQL 的 bin 目录，如图 10-24 所示。

（17）登录 MySQL，输入 mysql -u root -p，此命令意思是以 root 登录，并要求输入密码，输入密码后，界面如图 10-25 所示。

（18）登录到 MySQL 后，创建 Bugs 库，并创建使用该库的用户 Bugs，密码为 123456。具体命令方法如下。

- 建立数据库 Bugs。

```
create database bugs;
```

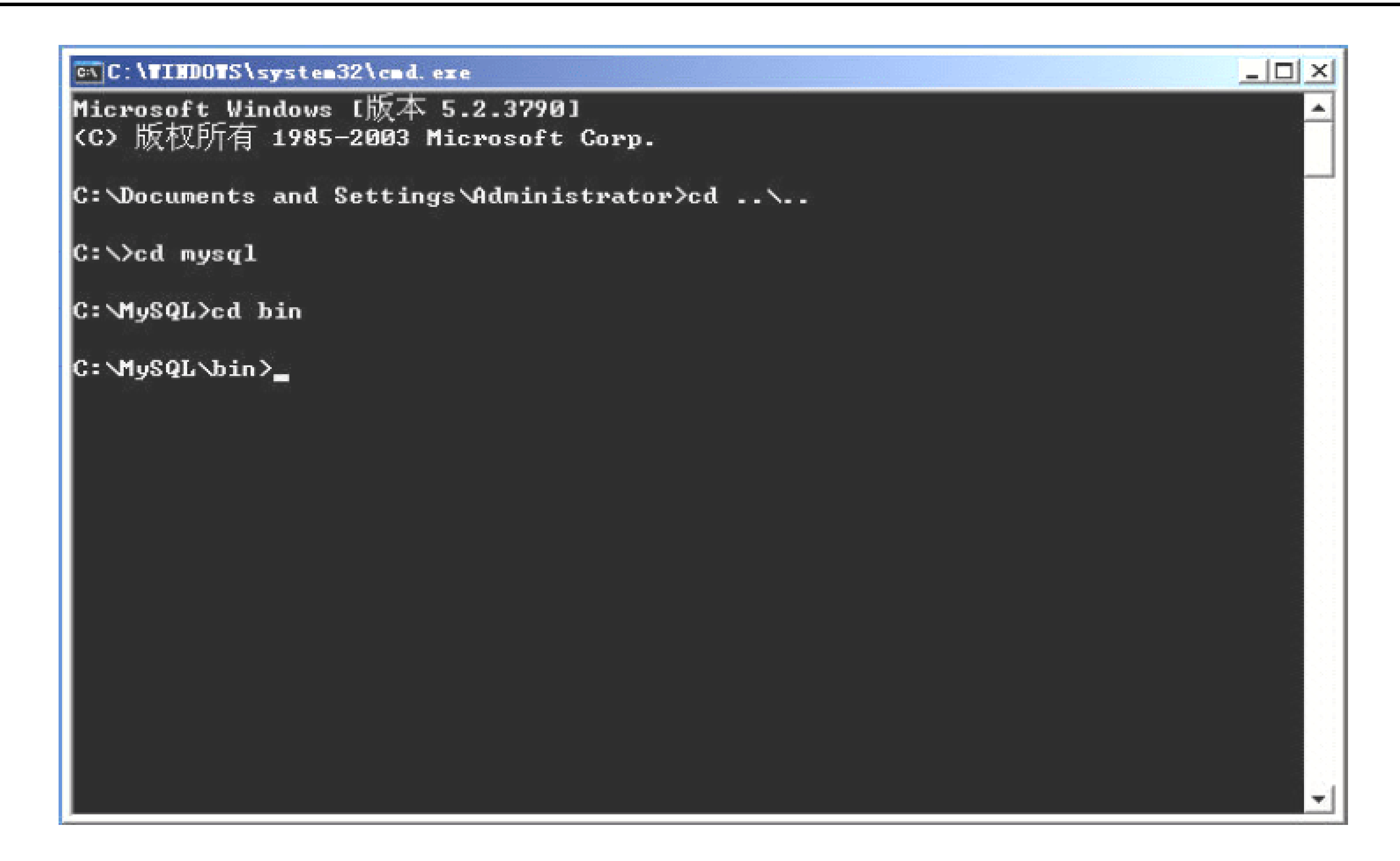



图 10-24　进入 MySQL bin 目录

```
C:\WINDOWS\system32\cmd.exe - mysql -u root -p
Microsoft Windows [版本 5.2.3790]
(C) 版权所有 1985-2003 Microsoft Corp.

C:\Documents and Settings\Administrator>cd ..\..

C:\>cd mysql

C:\MySQL>cd bin

C:\MySQL\bin>mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 5.0.18-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

图 10-25　登录 MySQL

• 建立用户 bugs，密码为空，对 Bugs 数据库授予一定权限。

```
GRANT SELECT,INSERT,UPDATE,DELETE,INDEX,ALTER,CREATE,LOCK TABLES,DROP,
REFERENCES ON bugs. * TO bugs@localhost IDENTIFIED BY  '123456';
```

其中 bugs@localhost 中的 bugs 为用户名，IDENTIFIED BY '123456'中的'123456'为 bugs 用户的密码。

• 从 MySQL 数据库授权表中重新装载权限。

```
FLUSH PRIVILEGES;
```

操作过程如图 10-26 所示。

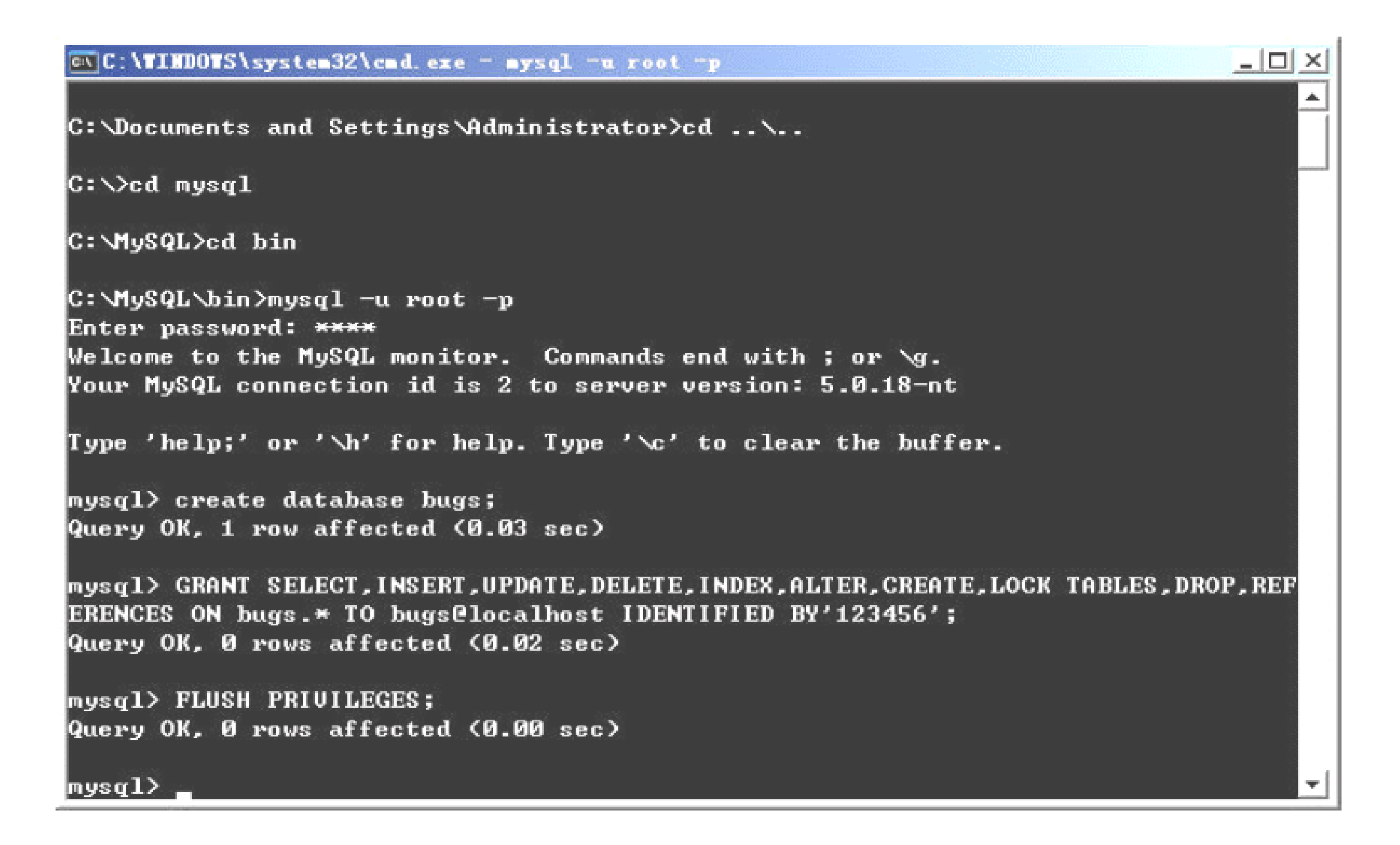



图 10-26　创建 bugs 数据及用户

### 3. 解压部署 Bugzilla

将 bugzilla-3.1.3.tar.gz 中的 bugzilla-3.1.3 解压放到 C 盘根目录下，并改名为 bugzilla，完成目录路径为 C：\bugzilla，如图 10-27 所示。

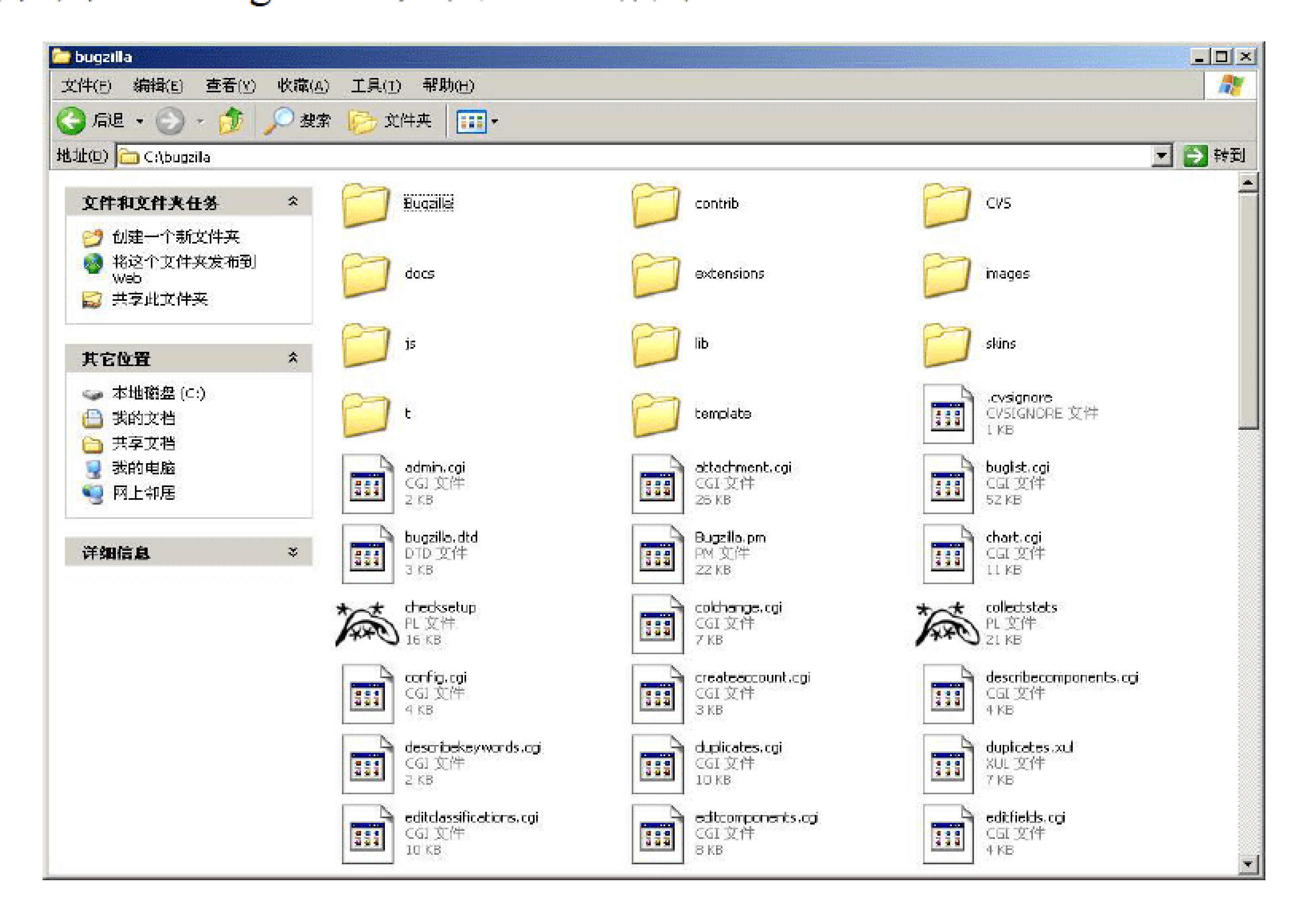



图 10-27　bugzilla 目录

### 4. 配置 IIS 服务

配置步骤

（1）打开“控制面板”/“管理工具”/“Internet 信息服务（IIS）管理器”，选择“默

认网站”，右击并选择“属性”，打开“主目录”，单击“配置”，在“应用程序扩展”中单击“添加”按钮，在“可执行文件”中输入 C：\Perl\bin\perl.exe –T“%s”%s，在“扩展名”中输入：.cgi，在“限制为”中输入：GET，HEAD，POST，单击“确定”按钮，完成 cgi 扩展添加。

在“应用程序扩展”中修改.pl 的可执行文件为 C：\Perl\bin\perl.exe –T“%s”%s，原来的是 C：\Perl\bin\perl.exe“%s”%s，其他不变。选择“默认网站”，右击并选择“新建”下的“虚拟目录”，出现图 10-28 所示的窗口，单击“下一步”按钮。




图 10-28 创建虚拟目录

（2）输入别名 bugzilla，单击“下一步”按钮，如图 10-29 所示。

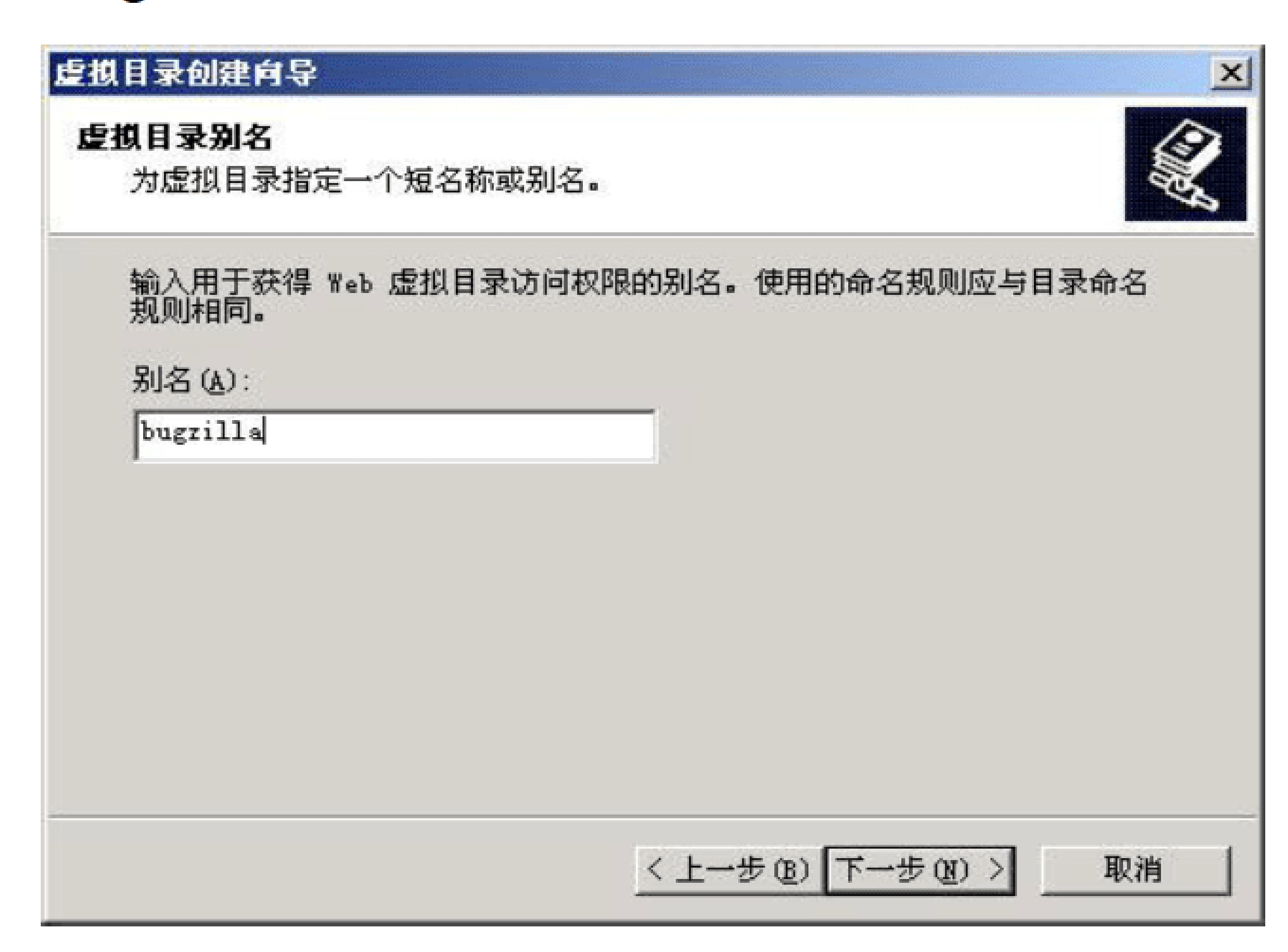



图 10-29 输入虚拟目录名称

（3）设置虚拟目录路径，这里选择 bugzilla 的目录，单击“下一步”按钮，如图 10-30 所示。

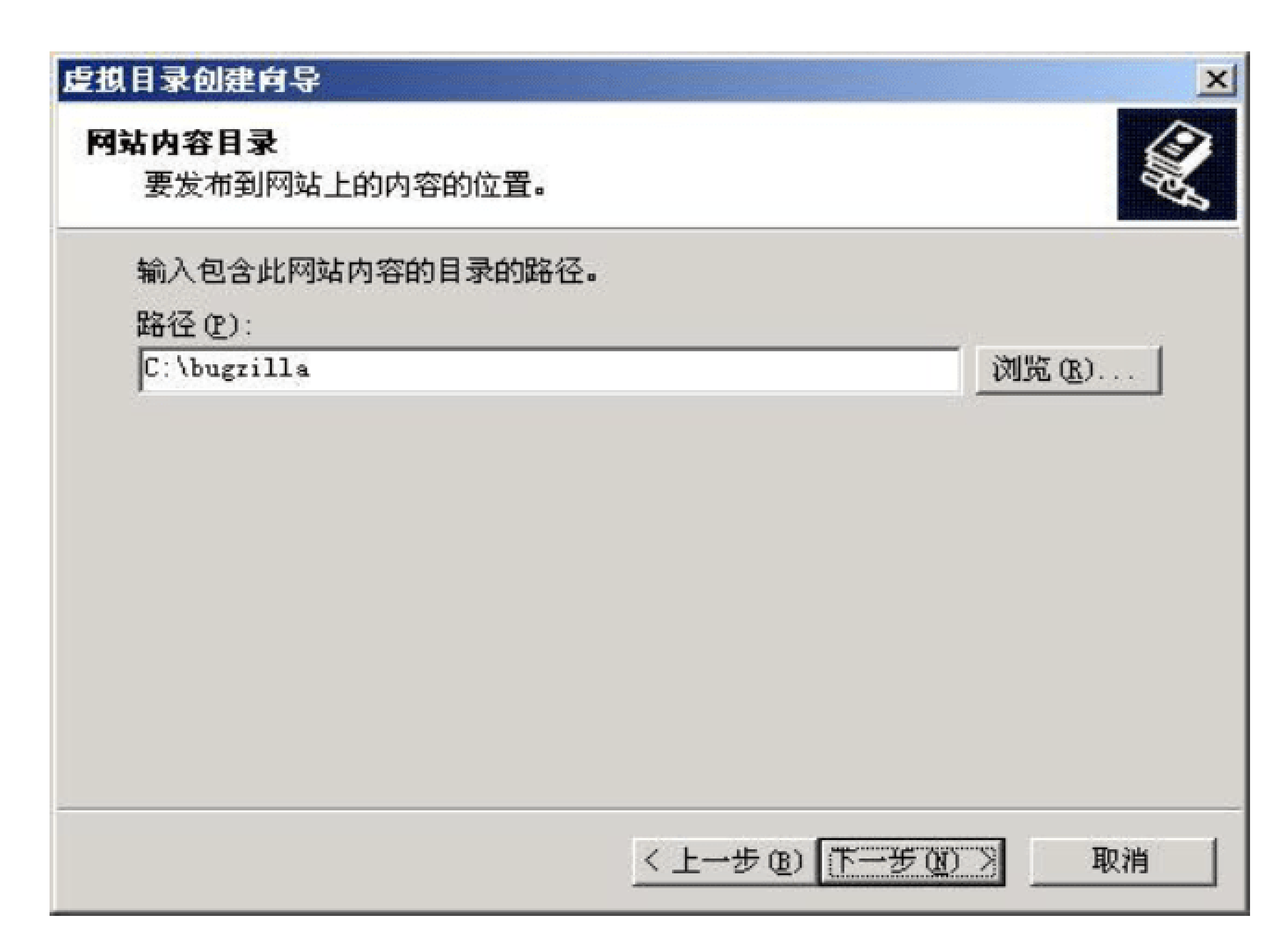



图 10-30 设置虚拟机目录路径

（4）按图 10-31 设置，单击“下一步”按钮，完成虚拟目录设置。

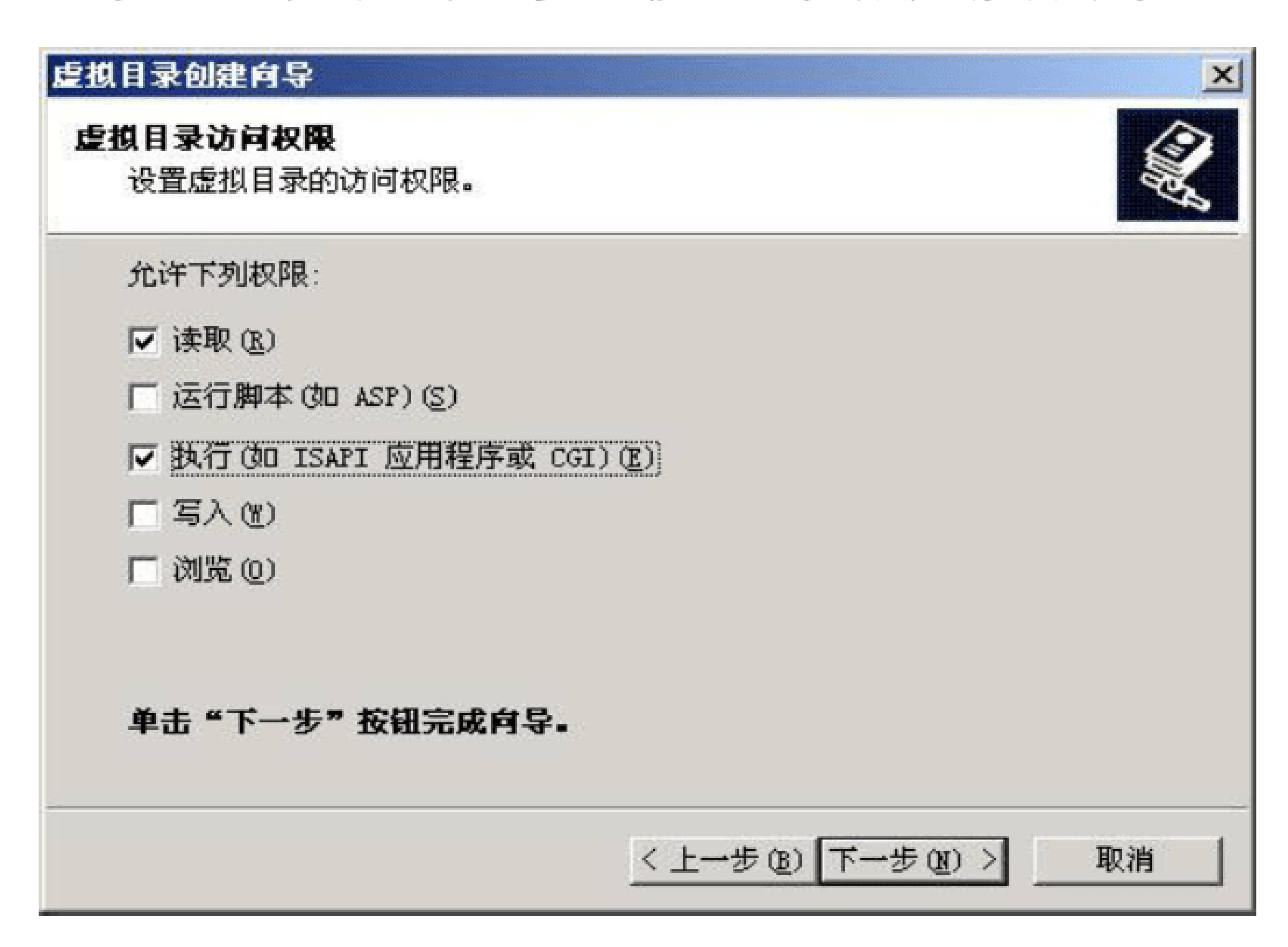



图 10-31 设置虚拟目录执行权限

（5）选择刚才建立的虚拟目录 bugzilla，右击并选择“属性”下的“文档”。在默认内容文档中增加 index.cgi。如图 10-32 所示。

（6）单击左窗格中的“Web 服务扩展”节点，启用相关扩展服务，如图 10-33 所示。

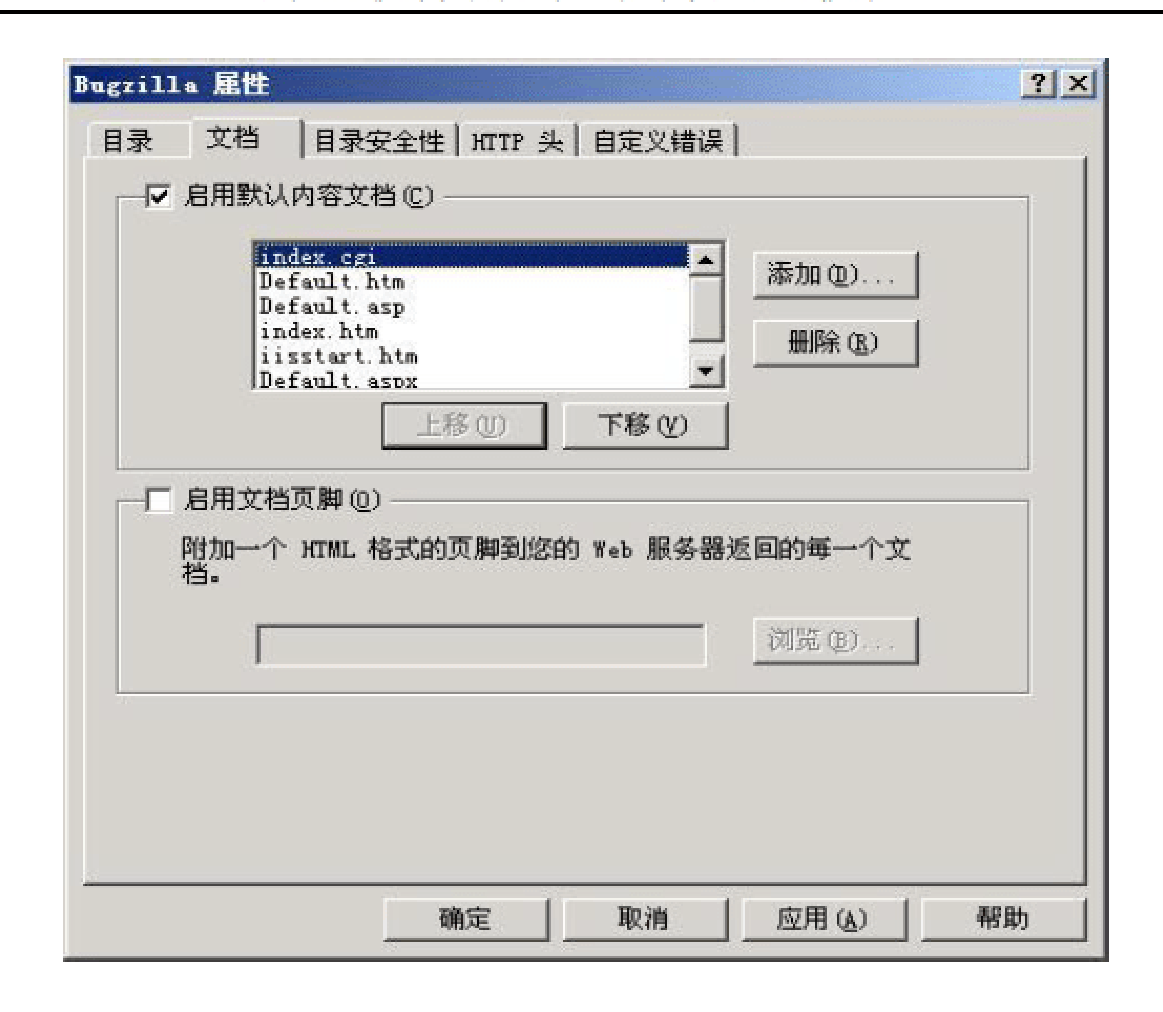



图 10-32　设置虚拟目录文档属性




图 10-33　Web 服务扩展

### 5．搭建邮件服务器

（1）安装 POP3 服务组件。

以系统管理员身份（Administrator）登录 Windows Server 2003 系统。依次进入“控制面板”|“添加或删除程序”|“添加 / 删除 Windows 组件”，在弹出的“Windows 组件向导”对话框中选中“电子邮件服务”选项，单击“详细信息”按钮，可以看到该选项包括两部分内容：POP3 服务和 POP3 服务 Web 管理。为方便用户远程 Web 方式管理邮件服务器，建议选中“POP3 服务 Web 管理”。

（2）安装 SMTP 服务组件。

选中“应用程序服务器”选项，单击“详细信息”按钮，接着在“Internet 信息服务

（IIS）”选项中查看详细信息，选中 SMTP Service 选项，最后单击“确定”按钮。此外，如果用户需要对邮件服务器进行远程 Web 管理，一定要选中“万维网服务”中的“远程管理（HTML）”组件。完成以上设置后，单击“下一步”按钮，系统就开始安装配置 POP3 和 SMTP 服务了。

（3）配置 POP3 服务器。

① 创建邮件域。单击“开始”|“管理工具”|“POP3 服务”，弹出 POP3 服务控制台窗口。选中左窗格中的 POP3 服务后，单击右窗格中的“新域”超链接，弹出“添加域”对话框，接着在“域名”栏中输入邮件服务器的域名，也就是邮件地址@后面的部分，如 symbio-group.com，最后单击“确定”按钮。如图 10-34 所示。

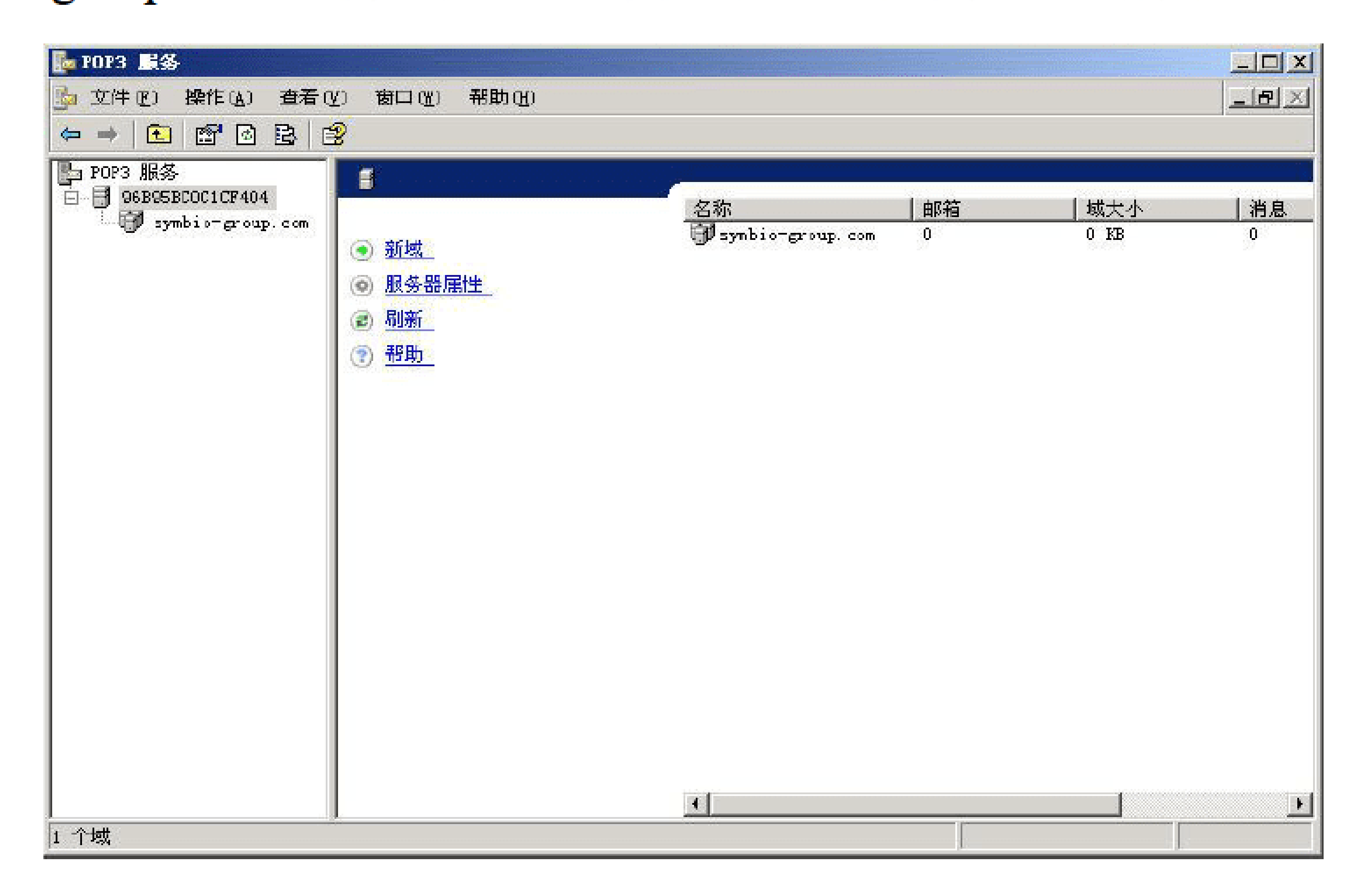



图 10-34　创建邮件域

② 创建用户邮箱。选中刚才新建的 symbio-group.com 域，在右窗格中单击“添加邮箱”，弹出添加邮箱对话框，在“邮箱名”栏中输入邮件用户名，然后设置用户密码，最后单击“确定”按钮，完成邮箱的创建。如图 10-35 所示。

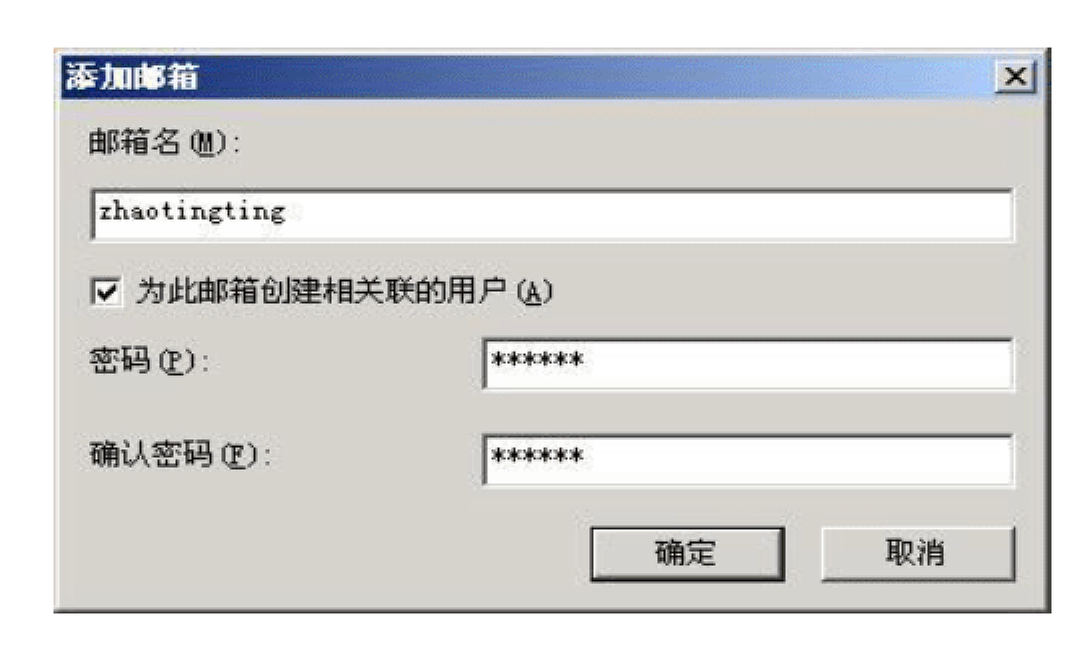



图 10-35　创建邮箱用户

创建成功界面如图 10-36 所示。

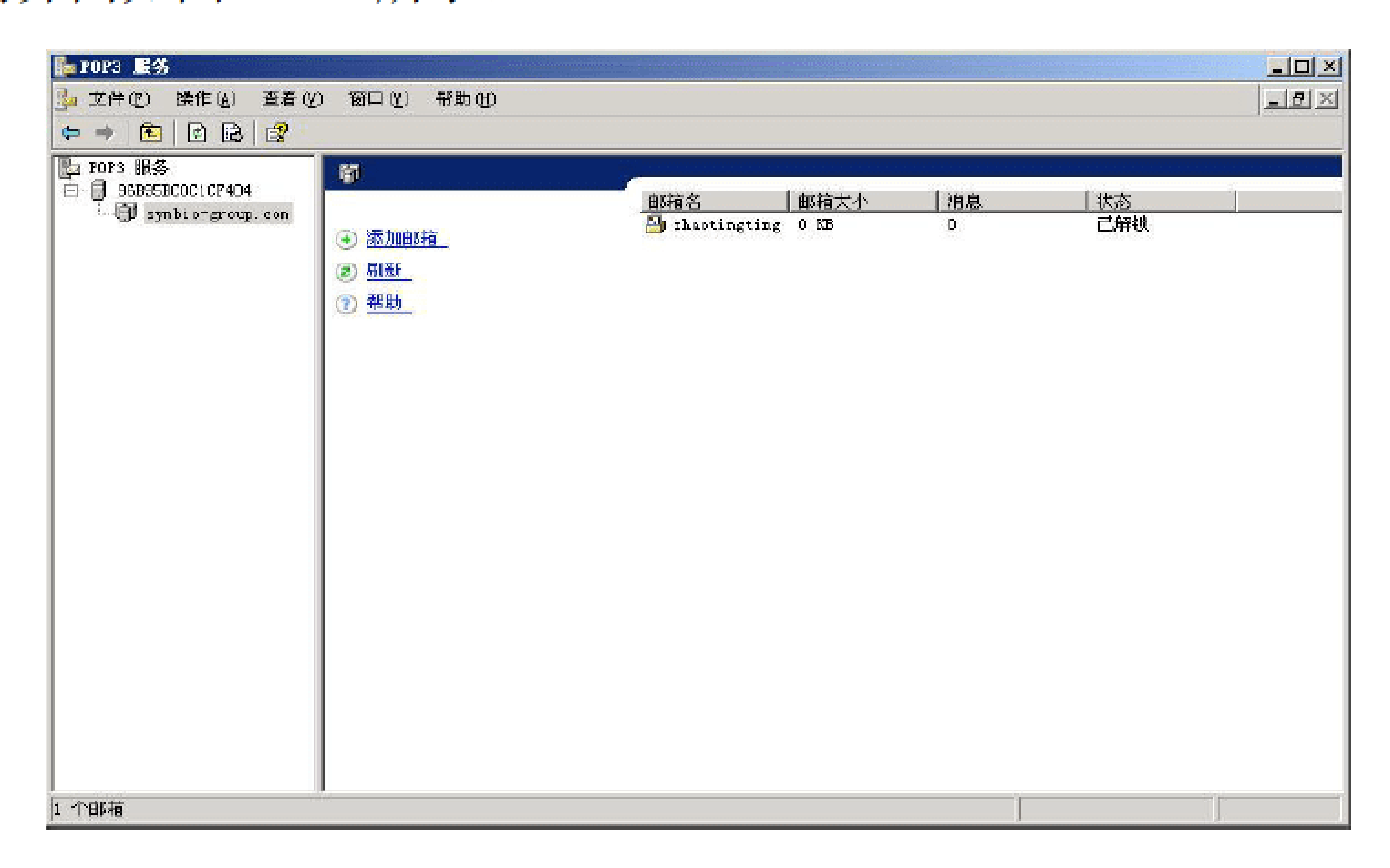



图 10-36 用户创建成功

③ 配置 SMTP 服务器。完成 POP3 服务器的配置后，就可开始配置 SMTP 服务器了。选择“开始”|“程序”|“管理工具”|“Internet 信息服务（IIS）管理器”，在“IIS 管理器”窗口中右击“默认 SMTP 虚拟服务器”选项，在弹出的菜单中选中“属性”，进入“默认 SMTP 虚拟服务器”窗口，如图 10-37 所示，切换到“常规”选项卡，在“IP 地址”下拉列表框中选中邮件服务器的 IP 地址即可。单击“确定”按钮，这样一个简单的邮件服务器就架设完成了。

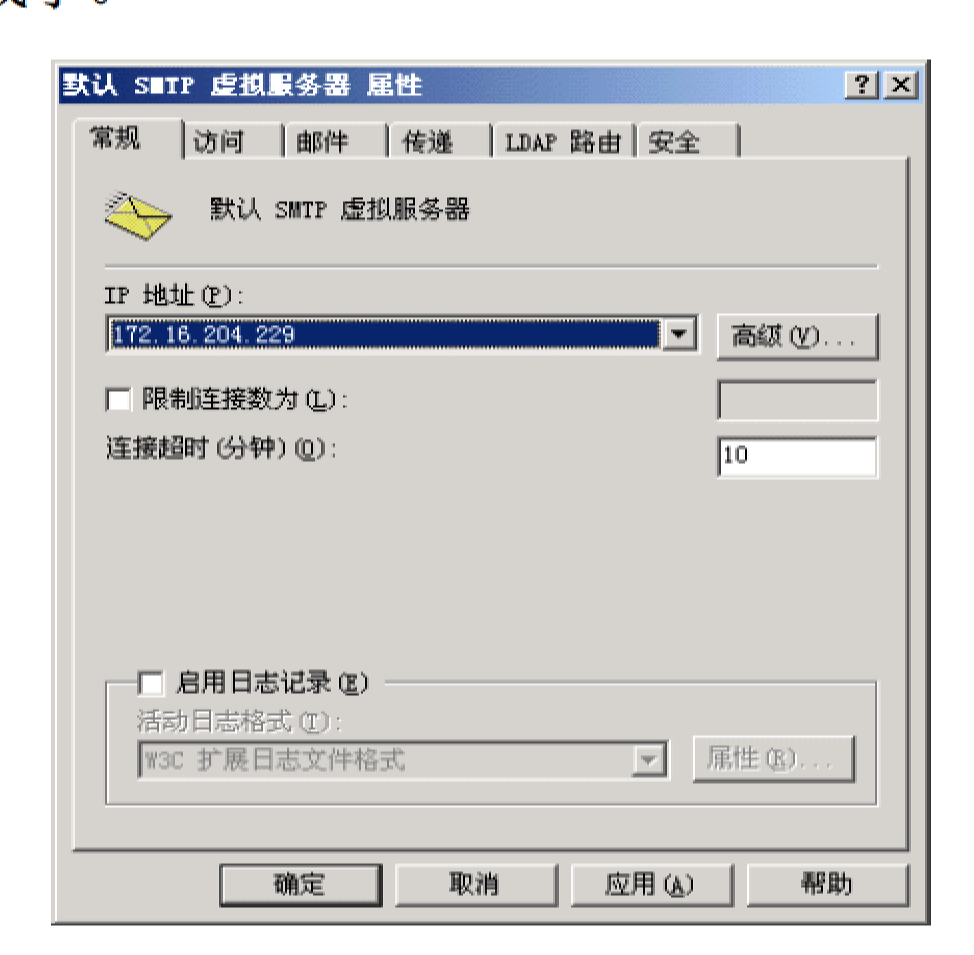



图 10-37 配置 SMTP 服务器

完成以上设置后，用户就可以使用邮件客户端软件连接邮件服务器进行邮件收发工作了。在设置邮件客户端软件的 SMTP 和 POP3 服务器地址时，输入邮件服务器的 IP 即

可。登录 Web 管理界面。

Windows Server 2003 还支持对邮件服务器的远程 Web 管理。在远端客户机中，运行 IE 浏览器，在地址栏中输入“https://服务器 IP 地址:8098”，将会弹出连接对话框，输入管理员用户名和密码，单击“确定”按钮，即可登录 Web 管理界面。

**6．安装 perl 模块**

上面步骤都正确无误后，我们开始 bugzilla 所需 perl 模块的安装。

安装步骤

（1）打开命令提示符，进入 bugzilla 目录，然后运行 checksetup.pl 检查所缺模块，如图 10-38 所示。

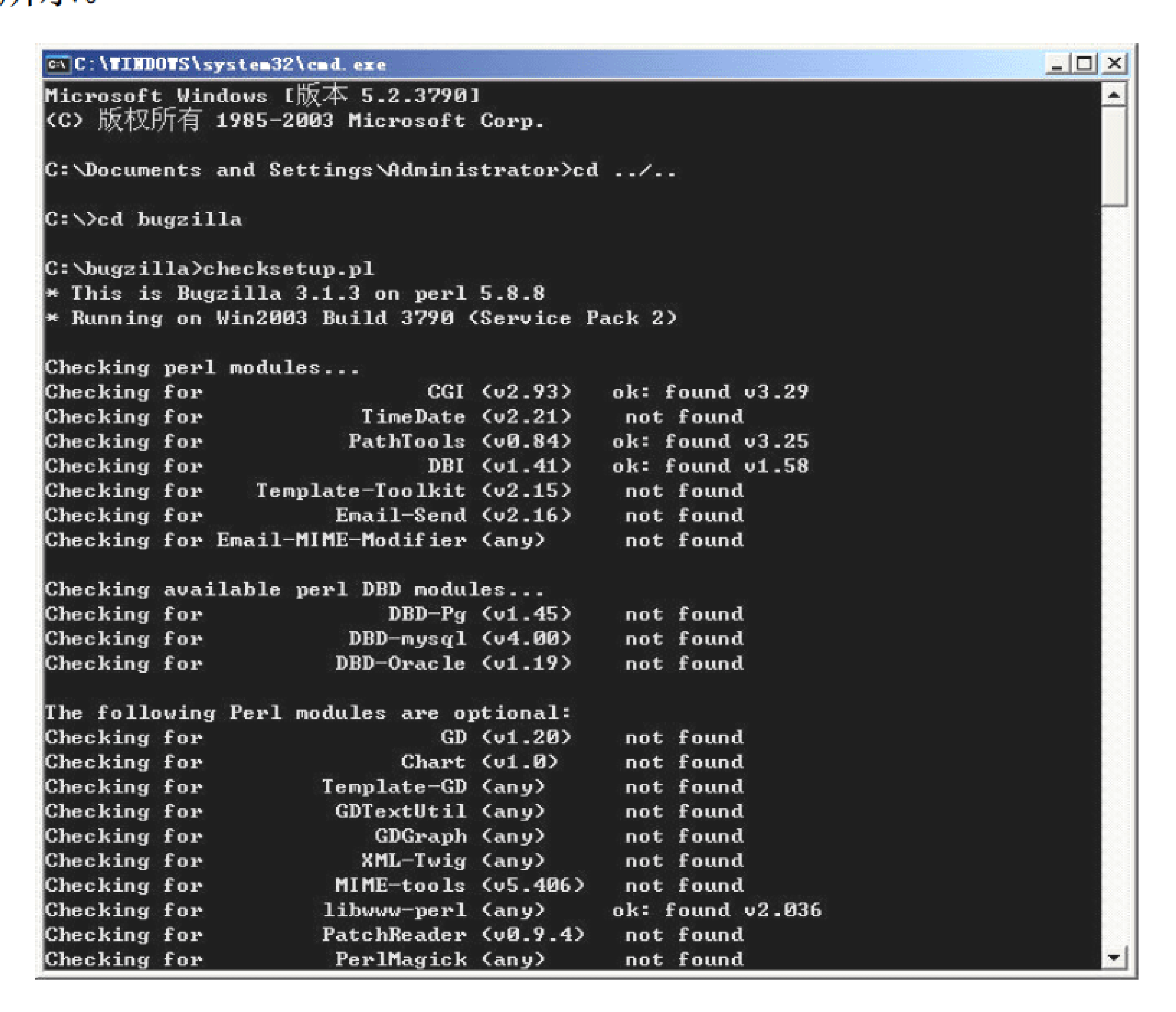



图 10-38　检查 perl 缺失模块

（2）如果缺少所需模块，一会显示 not found 字样，安装时，需将缺少的模块都安装上。在上面信息中，bugzilla 已经提示了缺少哪些文件，所有以 ppm 开头的都为缺少项，必须都安装，比如 ppm install TimeDate。右击命令提示符窗口，选择“标记”，然后选中需复制的信息，如 ppm install TimeDate，再单击右键，使用“粘贴”即完成命令输入，回车后进行安装。如图 10-39 所示。

在安装丢失模块之前，我们可以首先输入以下命令：

```
ppm repo add theory58S http://theoryx5.uwinnipeg.ca/ppms
```

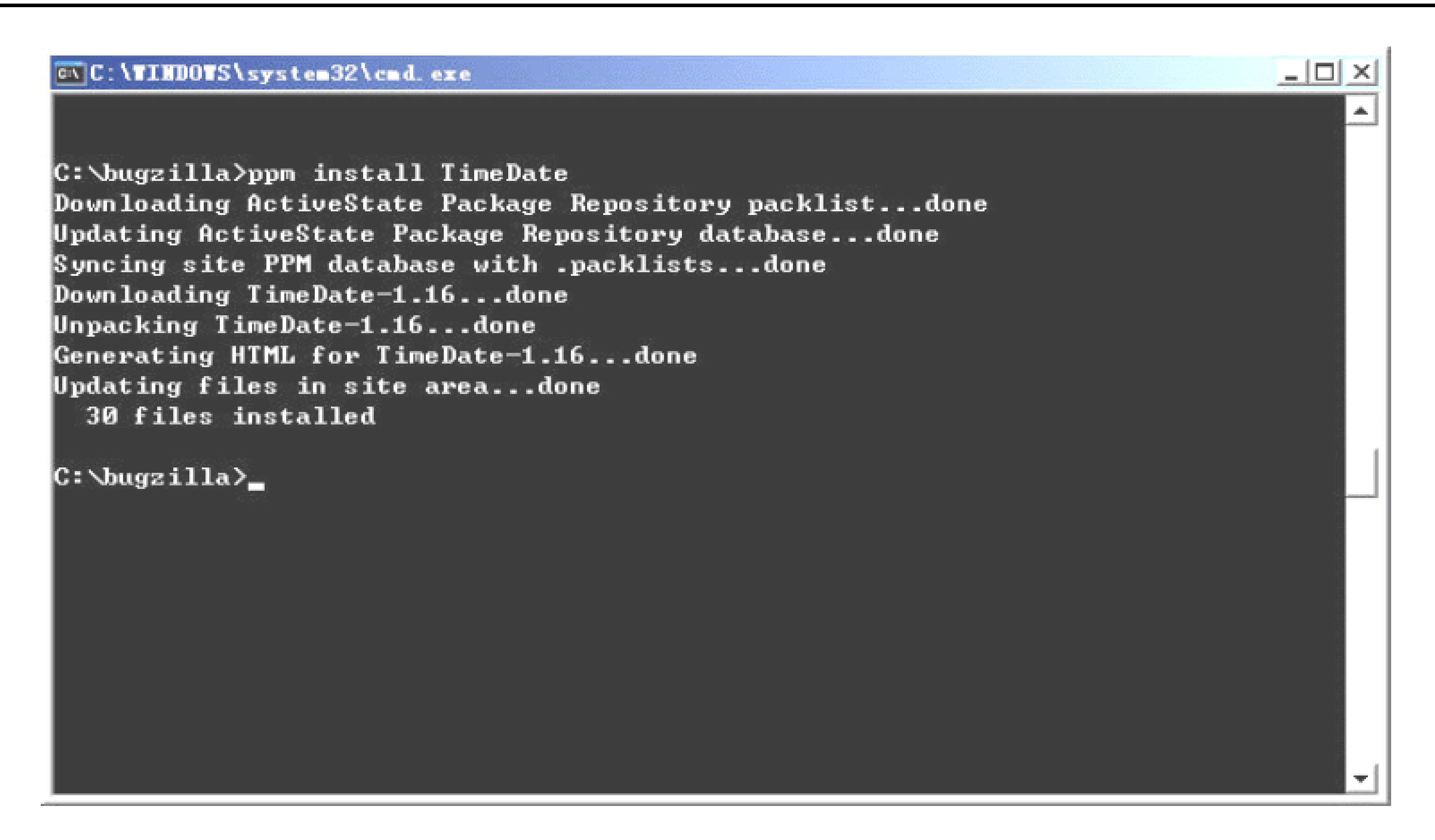



图 10-39 安装缺失 perl 模块

然后根据提示一步步安装 perl 模块。

**7. 修改 localconfig 文件**

所有模块安装完成后，再次运行 checksetup.pl，检查是否还有遗漏的模块未安装，如果没有，则提示要修改 localconfig 文件：

```
This version of Bugzilla contains some variables that you may want to
change and adapt to your local settings. Please edit the file
./localconfig and rerun checksetup.pl.

The following variables are new to ./localconfig since you last ran
checksetup.pl: create_htaccess, webservergroup, db_driver, db_host,
db_name, db_user, db_pass, db_port, db_sock, db_check, index_html, cvsbin,
interdiffbin, diffpath
```

进入 C：\bugzilla 中打开 localconfig，修改其中的数据库密码：

```
# The name of the database
$db_name = 'bugs';

# Who we connect to the database as.
$db_user = 'bugs';

# Enter your database password here. It's normally advisable to specify
# a password for your bugzilla database user.
# If you use apostrophe (') or a backslash (\) in your password, you'll
# need to escape it by preceding it with a '\' character. (\') or (\)
# (Far simpler just not to use those characters.)
$db_pass = '      ';---此处修改为实际的 bugs 用户密码，如上面设置'123456'.
```

修改完成后保存，再次在命令提示符中运行 checksetup．pl，如无问题，则开始相关信息的配置，具体配置内容如下列信息所示。

设置 SMTP 服务器，这里填上我们前面配置的邮件服务器地址：

```
Bugzilla requires an SMTP server to function on Windows.
Please enter your SMTP server's hostname: 172.16.204.229
```

设置一个超级管理员邮箱地址，即为 bugzilla 设置一个超级管理员：

```
Precompiling templates...done.
… …
Looks like we don't have an administrator set up yet.
Either this is your first time using Bugzilla, or your
administrator's privileges might have accidentally been deleted.

Enter the e-mail address of the administrator: zhaotingting@symbio-group.com
```

输入该超级管理员的真实姓名：

```
Enter the real name of the administrator: zhaotingting
```

设置超级管理员密码：

```
Enter a password for the administrator account: 123456
```

再次输入超级管理员密码：

```
Please retype the password to verify: 123456
```

其他配置信息：

```
zhaotingting@symbio-group.com is now set up as an administra
Creating default classification 'Unclassified'...
Creating initial dummy product 'TestProduct'...

Now that you have installed Bugzilla, you should visit the
'Parameters' page (linked in the footer of the Administrator
account) to ensure it is set up as you wish - this includes
setting the 'urlbase' option to the correct URL.
```

完成后，即可在 IE 中输入 http：//localhost / bugzilla 访问，如果配置无误，可见到图 10-40 所示的成功界面。

如果没有成功，请检查根据错误提示，检查上述步骤。接下来，我们可以做汉化的操作，汉化方法比较简单，只需将汉化包拷贝到 Bugzilla 所在目录的 template 下，然后

刷新 Bugzilla 页面即可完成汉化，汉化后的效果如图 10-41 所示。

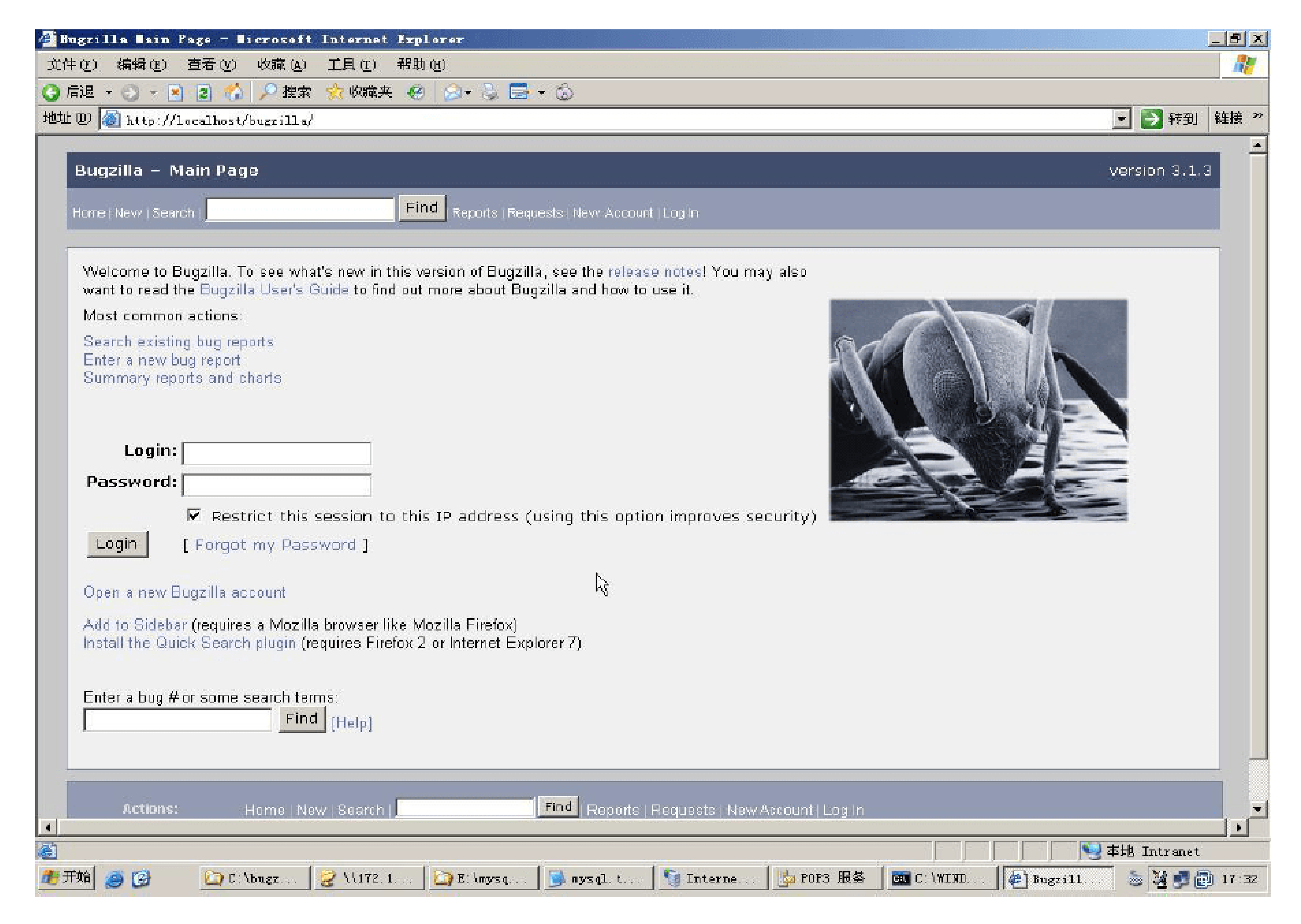



图 10-40 Bugzilla 系统界面




图 10-41 汉化后的 Bugzilla

# 10.3　缺陷数据库实例解析

## 10.3.1　报告软件缺陷

上面内容介绍了缺陷的定义，什么是软件缺陷，我们介绍如何在 Bugzilla 缺陷跟踪系统中提交缺陷报告，如何对提交的缺陷报告进行修改和查询如何对用户进行管理。登录 Bugzilla 缺陷跟踪系统的主页面如图 10-42 所示（以下是汉化之后的界面）。

我们用注册好的用户名和密码登录缺陷跟踪系统，登录完成后如图 10-43 所示。




图 10-42　登录 Bugzilla 缺陷跟踪系统主页面




图 10-43　用户名登录的主页面

接下来我们来学习如何提交缺陷报告，单击“新建”按钮，会弹出一个新的包括很多列表项的报告缺陷的界面，如图 10-44 所示。

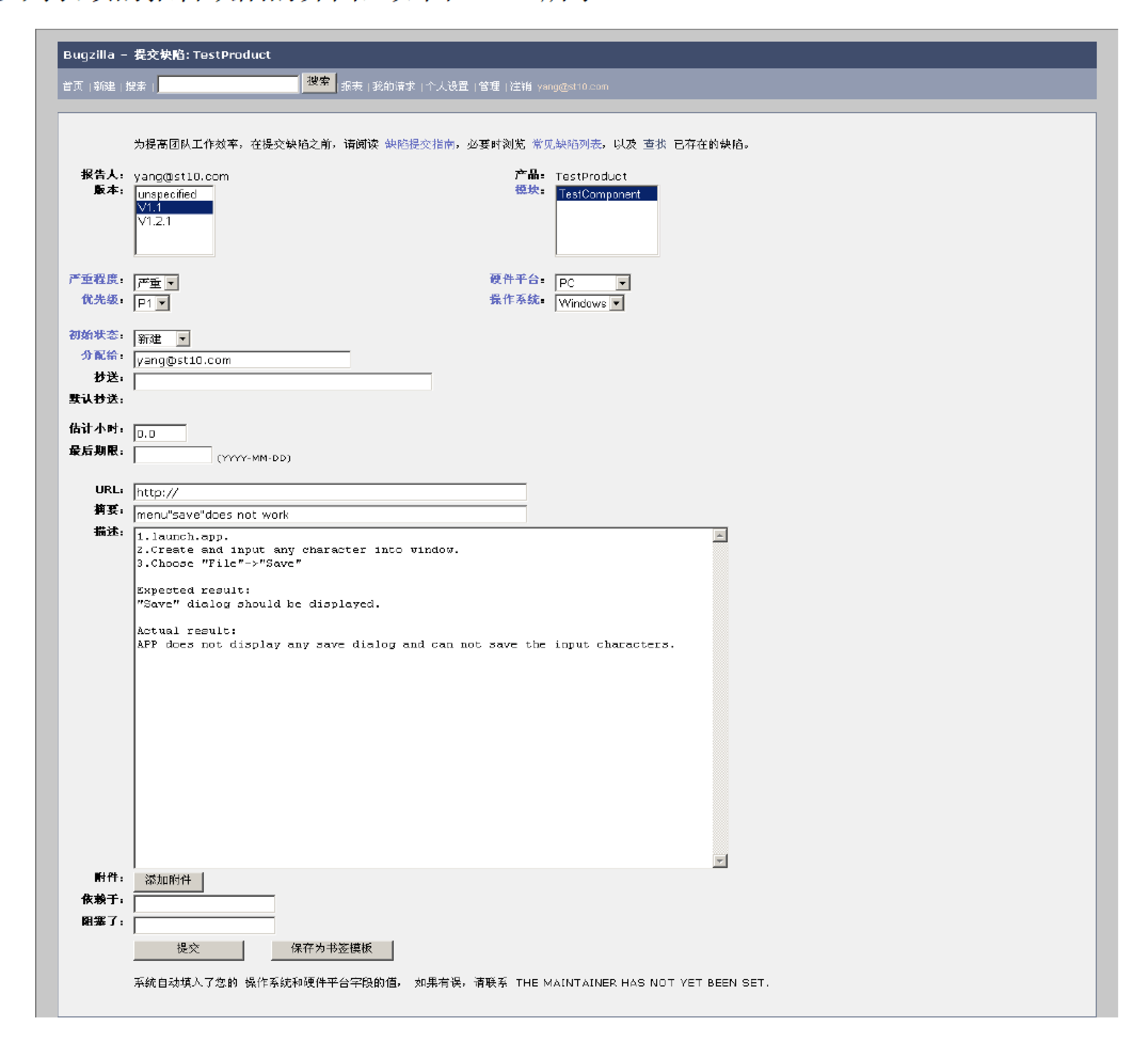



图 10-44　报告新缺陷的界面

（1）版本号：所要测试软件/系统的版本号。

（2）模块：表明存在缺陷的模块名称。

（3）严重程度：缺陷的类型。

（4）优先级：为该缺陷分配处理的优先级。

（5）硬件平台：硬件环境。

（6）操作系统：表示测试所用的操作系统的名称。

（7）初始状态：缺陷报告的初始状态。

（8）分配给：修复此缺陷的开发人员。

（9）抄送：与此缺陷报告相关的人员。

（10）估计时间：修复缺陷需要多长时间。

（11）最后期限：修改缺陷的最后时间。

（12）摘要：填写缺陷的标题，要求简明、准确和完整。

（13）描述：记录复现缺陷的详细操作步骤，以及“期望的结果”和“实际测试的结果”。

（14）附件：如果现象或结果文字表述不准确，通常以图片的形式作为附件上传，也可以上传任意类型的用于 Bug 重现的附件。

（15）依赖于：是指如果要修复当前 Bug，需要先修复 Bug 的 ID 号，或者表示当前 Bug 是由哪个 Bug 引起的，填写此 Bug 的 ID 号。

（16）阻塞了：是指如果不修复当前 Bug 那么此处填写的 ID 号的 Bug 也不能被修复。

输入完各项信息后之后，单击“提交”按钮则保存了当前的缺陷。缺陷被保存后，自动生成一个 ID 号，通过 ID 号，可以迅速查找某个缺陷。

## 10.3.2　编辑软件缺陷报告

缺陷报告已经提交，接下来的工作任务是对所提交的缺陷报告进行跟踪，那么就要编辑已打开的缺陷，可以在“产品管理”的界面如图 10-45 所示，单击缺陷数可以查看到所提交的 Bug 列表。如图 10-46 所示。

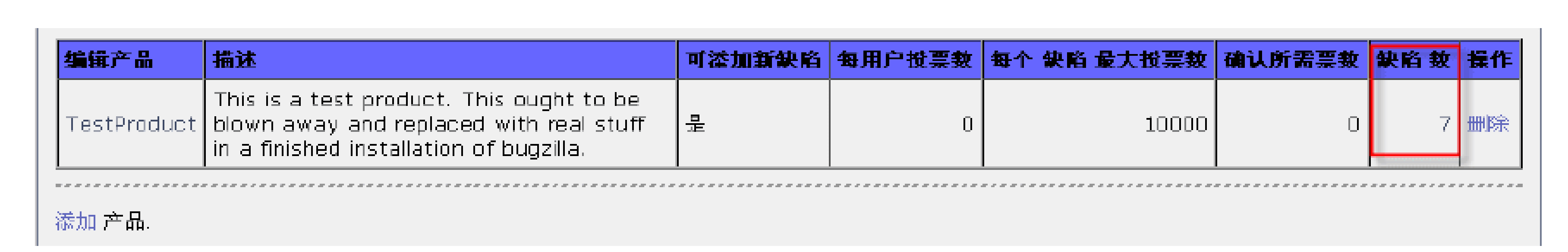



图 10-45　如何查看缺陷




图 10-46　查看缺陷之后的页面

单击要编辑的缺陷报告 ID 号，打开缺陷报告编辑界面，填写修改缺陷的一些信息，如图 10-47 所示。管理员单击“同时更改多个缺陷”可以同时修改多个缺陷报告。



图 10-47 修改缺陷报告

（1）添加抄送列表，将缺陷报告发送给修复此 Bug 的相关的人员。

（2）在“附加评论”文本框中输入修改缺陷的方法和缺陷产生的原因等文字，这有助于缺陷跟踪，对于代表性的缺陷和修正方法，可以供今后修正相同类型的缺陷时参考。

（3）在选择处理缺陷的方式，我们选择的状态为已解决。

最后，单击底部的“提交”按钮，保存缺陷的更新信息。

## 10.3.3 验证软件缺陷

缺陷被修正后，软件测试人员要在新的软件版本上验证。如果缺陷在新的版本上被修正了，则关闭缺陷，否则须再次打开缺陷，并更新缺陷的数据库信息。首先通过查询 ID 号为 7 的缺陷报告，如图 10-48 所示。输入 7，单击“搜索”按钮。




图 10-48　搜索选项

会出现 ID 为 7 的缺陷报告，如图 10-49 所示。

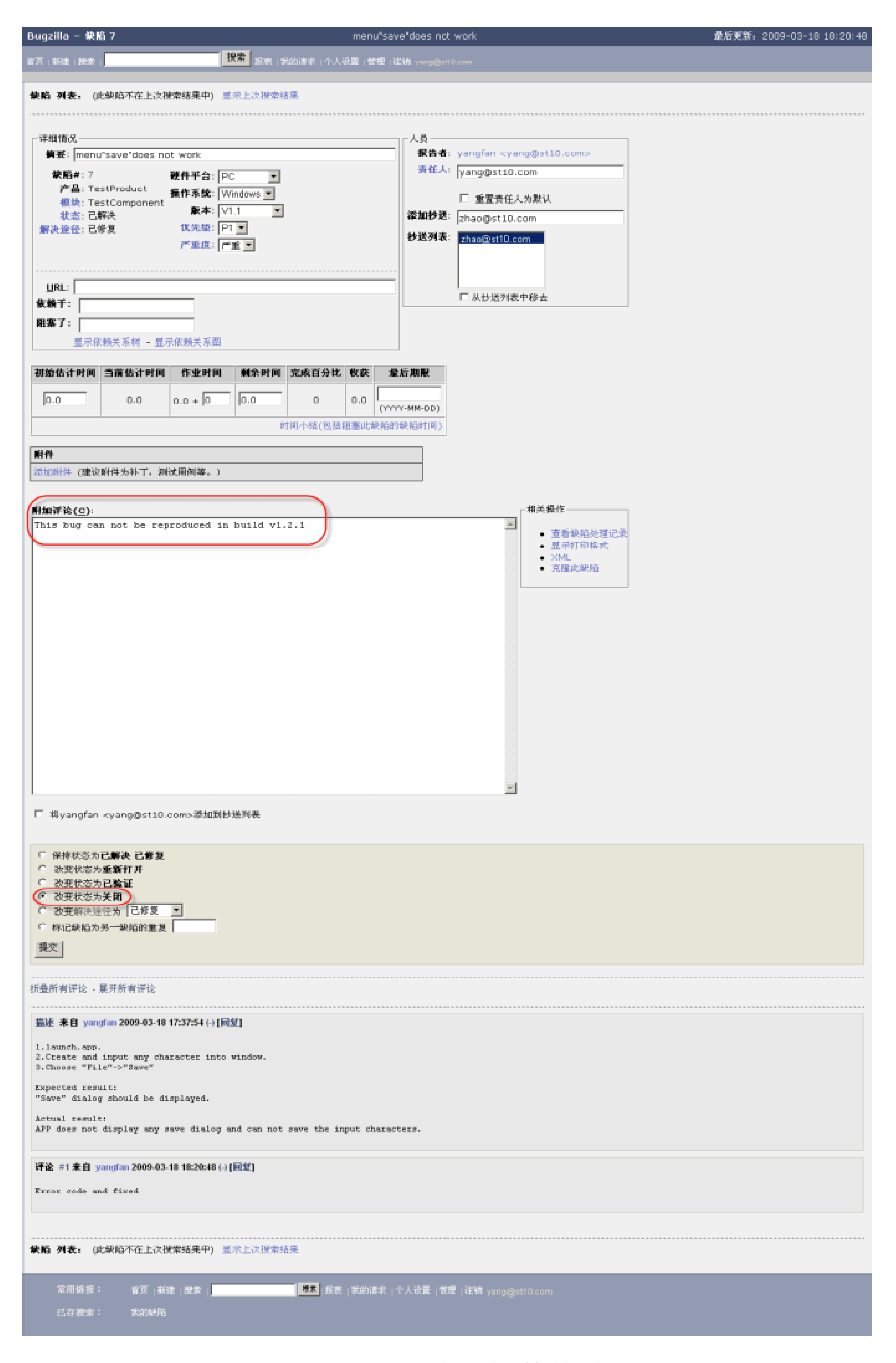



图 10-49　验证软件缺陷

如果缺陷被修正，则把处理状态改变为关闭。

“附加评论”中填写有关缺陷的重现、修复、补充和建议等信息。

最后，单击底部的“提交”按钮，保存缺陷的验证信息。

## 10.3.4 软件缺陷查询

缺陷搜索是经常使用的功能，缺陷跟踪数据库可以查找不同的字段包含的文字。用户可以在菜单栏单击“搜索”按钮打开搜索界面。要查找某个缺陷，可以通过如下的方式进行查询，我们可以通过搜索特定缺陷和高级搜索来进行软件缺陷报告的查询。如图 10-50 所示。

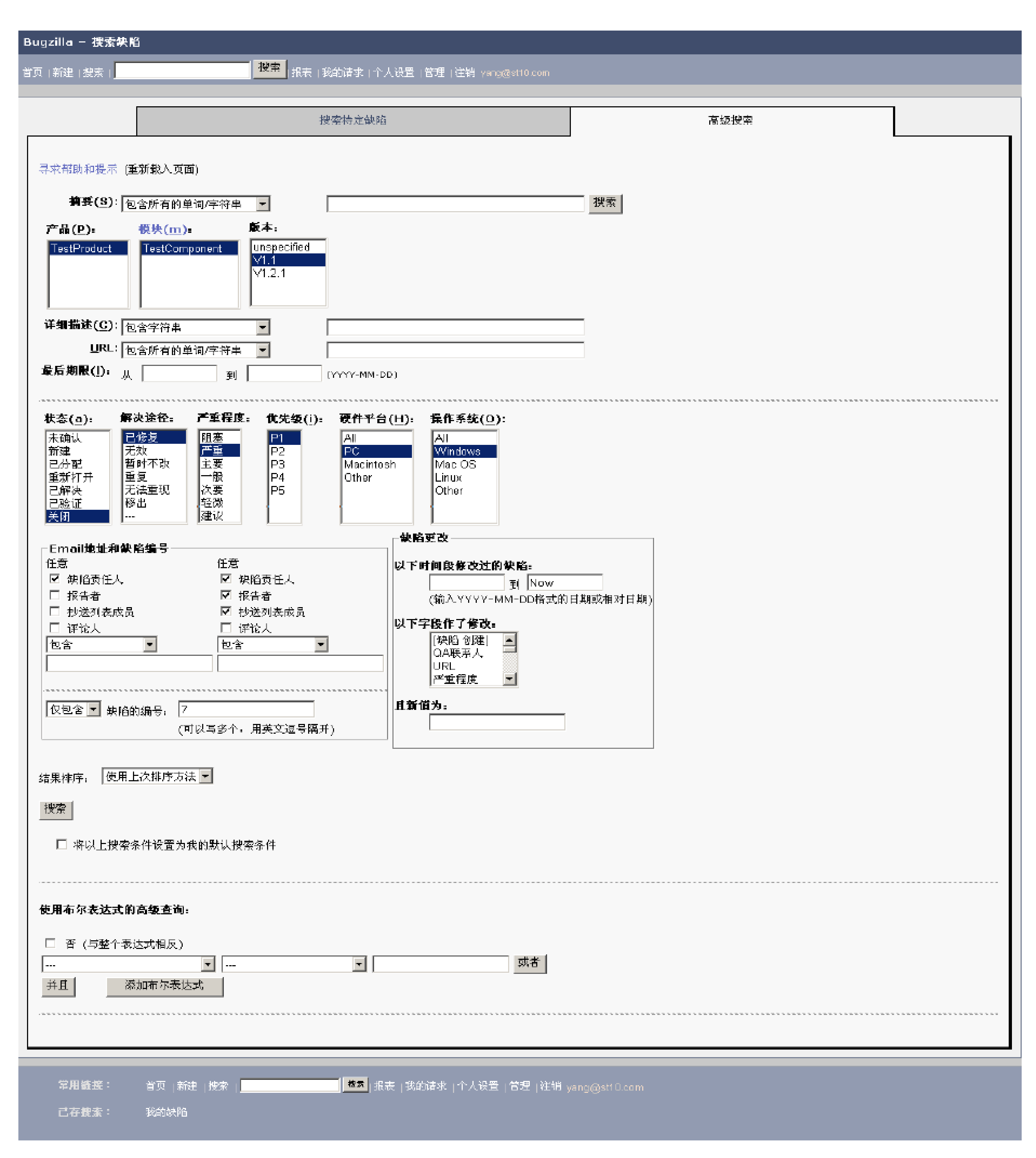



图 10-50 软件缺陷查询

选择的条件状态为“关闭”，解决途径为“已修复”严重程度为“严重”，优先级为“P1”，硬件平台为“PC”，操作系统为“Windows”来进行软件缺陷的查询。查询结果

如图 10-51 所示。

Bugzilla – 缺陷 列表

首页 | 新建 | 搜索 | 搜索 报表 | 我的请求 | 个人设置 | 管理 | 注销 yang@st10.com

Wed Mar 18 2009 18:56:56

此处为放置随机标语的地方，但目前系统中无任何标语可用。

| ID | 严重级 | 优先级 | 操作系统 | 责任人 | 状态 | 解决途径 | 摘要 |
|---|---|---|---|---|---|---|---|
| 7 | 严重 | P1 | Window | yang@st10.com | 关闭 | 已修复 | menu"save"does not work |

找到一个缺陷。

长格式

XML　CSV | Feed | iCalendar | 改变显示列 | 编辑搜索条件　保存搜索 为

作业时间概述

在产品 "TestProduct" 中录入新缺陷

图 10-51　缺陷的查询结果

## 10.3.5　注册用户管理

缺陷报告提交之后，下面介绍用户管理这方面的知识，首先通过以下方式进入用户管理界面。

第一步，单击菜单栏“管理”，如图 10-52 所示。




图 10-52　菜单栏的“管理”按钮

第二步，在图 10-53 所示的用户首页单击“用户管理”。打开用户管理界面，如图 10-54 所示。




图 10-53　用户首页

Bugzilla 缺陷跟踪系统为管理员提供了添加、修改和删除注册用户的功能。




图 10-54　用户管理界面

### 1．查看所有用户

我们可以查到注册用户的相关信息。直接单击“搜索”按钮可以查看到所有用户的列表，如果 10-55 所示。


已找到2个用户。

| 编辑用户 | 真实姓名 | 帐号历史 |
| --- | --- | --- |
| yang@st10.com | yangfan | 查看 |
| zhao@st10.com | zhaoting | 查看 |

如果您此时不想修改用户帐号，您可以 搜索其它用户 或 添加新用户。

图 10-55　用户列表

### 2．查询特定用户

如果要查找特定用户可以在“搜索文本框”中输入特定用户的关键字，然后单击“搜索”按钮，如图 10-56 所示。

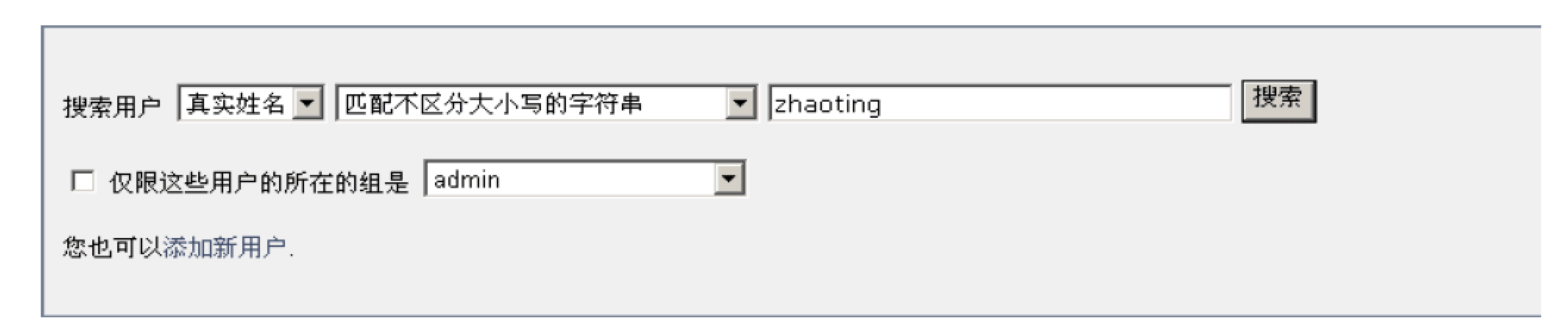



图 10-56　搜索特定用户

搜索特定用户的信息结果如图 10-57 所示。

### 3．编辑用户

在图 10-57 所示的界面单击“查看”|“编辑此用户”可对此用户的信息进行修改。管理员可以给用户相应的权限将其分配给不同组，如图 10-58 所示。

已找到1个用户。

| 编辑用户 | 真实姓名 | 帐号历史 |
| --- | --- | --- |
| zhao@st10.com | zhaoting | 查看 |

如果您此时不想修改用户帐号，您可以 搜索其它用户 或 添加新用户。

图 10-57　搜索特定用户的结果

用户名： zhao@st10.com

真实姓名： zhaoting

密码：
(要修改密码，请在此处输入新密码)

禁用缺陷邮件： (这只会对缺陷邮件和订阅邮件起作用，不会对重设密码或其它非缺陷相关邮件起作用)

禁用说明：
(如果非空，当此帐号被禁用时，这段文字将解释禁用原因。)

组权限　可授予其它用户权限
| 用户是以下组的成员

[☑] ☑ admin: Administrators
[ ] [ ] bz_canusewhineatothers: Can configure whine reports for other users
[ ] [ ] bz_canusewhines: User can configure whine reports for self
[ ] [ ] bz_sudoers: Can perform actions as other users
[ ] [ ] bz_sudo_protect: Can not be impersonated by other users
[ ] [ ] canconfirm: Can confirm a bug or mark it a duplicate
[ ] [ ] creategroups: Can create and destroy groups
[ ] * [ ] * editbugs: Can edit all bug fields
[ ] [ ] editclassifications: Can create, destroy, and edit classifications
[ ] [ ] editcomponents: Can create, destroy, and edit components
[ ] [ ] editkeywords: Can create, destroy, and edit keywords
[ ] [ ] editusers: Can edit or disable users
[ ] [ ] tweakparams: Can change Parameters

产品职责： 无

保存修改 或查看帐号历史

图 10-58　编辑用户界面

## 4．添加用户

单击“添加新用户”可以进入添加用户界面，如图 10-59 所示。

输入相关信息，进行用户的添加。

- 用户名：输入新建的用户名。




图 10-59　添加用户界面

- 真实姓名：输入用户的真实姓名。
- 密码：输入用户密码。

# 第 11 章　如何成为合格的外包软件测试工程师

**学习目标：**

1．掌握外包软件测试工程师的基本素质
2．理解软件测试的工作经验和技巧
3．确定自己的职业发展方向

## 11.1　国内外包软件测试工程师现状

从企业方面看，很多公司很难招聘到满意的测试工程师，另外现在人才问题成了软件测试外包企业的发展瓶颈，这说明整个测试行业的发展遇到了瓶颈问题。

从个人方面看，很多测试人员薪资和职位到了一定阶段就不再发展，例如很多测试工程师做到测试经理后就不能进一步发展。

就整个 IT 行业来看，尤其是与开发人员相比，测试工程师的待遇显得很低。就现在掌握的资料来看，同一级别的开发工程师要比测试工程师高。而测试工程师的职业发展空间主要如下。

（1）测试组长：这类测试人员通常是测试项目的负责人，既要具备较高的测试技术能力，还要具备一定的管理能力。主要职责是制订测试计划、编写测试计划、监控和管理整个测试过程。测试组长可以向上发展为测试部经理、质量经理，也可以横向发展为项目经理，而且通常待遇相对较高些。

（2）测试分析师：主要职责是对系统的测试结果进行综合的分析，例如缺陷分析、性能分析等。测试分析师不但测试技术能力较强，还要具备数据库、操作系统等多方面的技术知识。这类职务的发展空间也不错，可以发展成系统设计师等。

（3）自动化测试工程师、测试开发工程师：主要职责是编写测试程序、执行自动化测试任务。这类职位的测试人员至少要达到初级程序员的能力，因为经常和程序打交道。发展空间也不错，例如可以发展为程序员。

## 11.2　做一名合格的外包软件测试工程师

很多年轻或者刚刚从事测试工作的工程师，经常会问："测试工程师需要什么技能或者具有什么素质才是合格的？"与开发人员相比，测试人员不但需要一技之长，还需要掌握诸如操作系统、数据库、网络等多方面的知识。

经过这几年的发展，国内 IT 公司的测试水平有了很大的提高，但是与此同时，很多测试工程师也迎来了个人的发展瓶颈：很多人从测试工程师做到了测试经理的职位，不知道下一步如何发展，或者每天机械地从事着功能测试工作。

一个有竞争力的测试人员要具有下面三个方面的素质。

## 11.2.1 计算机专业技能

计算机领域的专业技能是测试工程师应该必备的一项素质，是做好测试工作的前提条件。尽管没有任何 IT 背景的人也可以从事测试工作，但是一名要想获得更大发展空间或者持久竞争力的测试工程师，则计算机专业技能是必不可少的。计算机专业技能主要包含三个方面。

**1. 测试专业技能**

现在软件测试已经成为一个很有潜力的专业。要想成为一名优秀的测试工程师，首先应该具有扎实的专业基础，这也是本书的编写目的之一。因此，测试工程师应该努力学习测试专业知识，告别简单的“点击”之类的测试工作，让测试工作以自己的专业知识为依托。

测试专业知识很多，本书内容主要以测试人员应该掌握的基础专业技能为主。测试专业技能涉及的范围很广，既包括黑盒测试、白盒测试、测试用例设计等基础测试技术，也包括单元测试、功能测试、集成测试、系统测试、性能测试等测试方法，还包括基础的测试流程管理、缺陷管理、自动化测试技术等知识。

**2. 软件编程技能**

“测试人员是否需要编程？”可以说是测试人员最常提出的问题之一。实际上，由于在我国开发人员待遇普遍高于测试人员，因此能写代码的几乎都去做开发了，而很多人则是因为做不了开发或者不能从事其他工作才“被迫”从事测试工作。最终的结果则是很多测试人员只能从事相对简单的功能测试，能力强一点的则可以借助测试工具进行简单的自动化测试（主要录制、修改、回放测试脚本）。

软件编程技能实际应该是测试人员的必备技能之一，在微软，很多测试人员都拥有多年的开发经验。因此，测试人员要想得到较好的职业发展，必须能够编写程序。只有能够编写程序，才可以胜任诸如单元测试、集成测试、性能测试等难度较大的测试工作。

此外，对软件测试人员的编程技能要求也有别于开发人员：测试人员编写的程序应着眼于运行正确，同时兼顾高效率，尤其体现在与性能测试相关的测试代码编写上。因此测试人员要具备一定的算法设计能力。依据资深测试工程师的经验，测试工程师至少应该掌握 Java、C#、C++之类的一门语言以及相应的开发工具。

**3. 网络、操作系统、数据库、中间件等知识**

与开发人员相比，测试人员掌握的知识具有“博而不精”的特点，“艺多不压身”是个非常形象的比喻。由于测试中经常需要配置、调试各种测试环境，而且在性能测试中

还要对各种系统平台进行分析与调优，因此测试人员需要掌握更多网络、操作系统、数据库等方面知识。

在网络方面，测试人员应该掌握基本的网络协议以及网络工作原理，尤其要掌握一些网络环境的配置，这些都是测试工作中经常遇到的知识。

操作系统和中间件方面，应该掌握基本的使用以及安装、配置等。例如很多应用系统都是基于 UNIX、LINUX 来运行的，这就要求测试人员掌握基本的操作命令以及相关的工具软件。而 WebLogic、WebSphere 等中间件的安装、配置很多时候也需要掌握一些。

数据库知识则是更应该掌握技能，现在的应用系统几乎离不开数据库。因此不但要掌握基本的安装、配置，还要掌握 SQL。测试人员至少应该掌握 MySQL、MS Sqlserver、Oracle 等常见数据库的使用。

作为一名测试人员，尽管不能精通所有的知识，但要想做好测试工作，应该尽可能地去学习更多的与测试工作相关的知识。

### 11.2.2　行业知识

行业主要指测试人员所在企业涉及的行业领域，例如很多 IT 企业从事石油、电信、银行、电子政务、电子商务等行业领域的产品开发。行业知识即业务知识，是测试人员做好测试工作的又一个前提条件，只有深入地了解了产品的业务流程，才可以判断出开发人员实现的产品功能是否正确。

很多时候，软件运行起来没有异常，但是功能不一定正确。只有掌握了相关的行业知识，才可以判断出用户的业务需求是否得到了实现。

行业知识与工作经验有一定关系，通过时间即可以完成积累。

### 11.2.3　个人素养

作为一名优秀的测试工程师，首先要对测试工作有兴趣，测试工作很多时候都是显得有些枯燥的，因此热爱测试工作，才更容易做好测试工作。因此，除了具有前面的专业技能和行业知识外，测试人员应该具有一些基本的个人素养，即下面的“五心”。

- 专心，主要指测试人员在执行测试任务的时候要专心，不可一心二用。经验表明，高度集中精神不但能够提高效率，还能发现更多的软件缺陷，业绩最棒的往往是团队中做事精力最集中的那些成员。
- 细心，主要指执行测试工作时候要细心，认真执行测试，不可以忽略一些细节。某些缺陷如果不细心很难发现，例如一些界面的样式、文字等。
- 耐心，很多测试工作有时候显得非常枯燥，需要很大的耐心才可以做好。如果比较浮躁，就不会做到“专心”和“细心”，这将让很多软件缺陷从你眼前逃过。
- 责任心，责任心是做好工作必备的素质之一，测试工程师更应该将其发扬光大。如果测试中没有尽到责任，甚至敷衍了事，这将会把测试工作交给用户来完成，

很可能引起非常严重的后果。

- 自信心，自信心是现在多数测试工程师都缺少的一项素质，尤其在面对需要编写测试代码等工作的时候，往往认为自己做不到。要想获得更好的职业发展，测试工程师们应该努力学习，建立能“解决一切测试问题”的信心。

“五心”只是做好测试工作的基本要求，测试人员应该具有的素质还很多。例如测试人员不但要具有团队合作精神，而且应该学会宽容待人，学会去理解“开发人员”，同时要尊重开发人员的劳动成果——开发出来的产品。

## 11.3 职业经验

### 11.3.1 职业发展

与国外的专业测试工程师相比，国内测试工程师突出特点是晋升非常快，2～3 年可能走完了国外 10 年的路。实际上有很多测试经理也就 2～3 年的工作经验，而在国外，测试工程师很多至少会有 5 年以上的开发经验。

国内的测试工程师目前的发展和国外有些类似，基本遵循从初级测试工程师到中级、高级测试工程师，再从测试负责人到测试经理、技术总监的发展历程。测试工程师如果把测试作为自己的职业，就应该充分了解上面的各个阶段的要求，同时在每个阶段打好基础。当然这个阶段的划分也不是严格的，最重要的还是看个人的努力程度。

可以看成，国内的测试行业发展很不正常，遇到职业发展瓶颈是非常合理的。我们可以先借鉴国外测试工程师的职业发展历程，很多时候，国内的软件都是跟着国外的发展趋势，测试行业国外至少比国内起步早 15～20 年。测试工程师要想真正地发展好自己、在市场上有长久的核心竞争力，就应该很好地规划自己的人生发展。

通常国外测试工程师的职业发展历程分为以下几个阶段：初级测试工程师、测试工程师/程序分析员、高级测试工程师/程序分析员、测试组负责人、测试/编程负责人、测试/质量保证/开发（项目）经理、计划经理。

我们用以下几个小案例来进行分析。

**案例 1：踏实地发展自己**

A 在北京工作有 4 年了。职业发展依次经历了测试员→测试工程师→测试分析师→测试经理。这就是 A 在北京的 4 年测试生涯。A 对测试工作有如下的行业观点：

① 软件测试不像一些人看起来那么简单，需要相当深厚的技术背景。但只要掌握要领，并不像我们普遍认为的那么困难。

② 测试工程师和开发人员相比，有机会接触更多的不同行业的项目，这是优势。

③ 测试工程师要想成功，更多的是靠平时的积累。不管是项目的积累，还是平时学习，两者都至关重要。

④ 测试工程师要充分利用网络资源，与同行们充分交流，在互相帮助和学习的氛围中，可以加快自己的成长速度。

> 点评：这是一位比较踏实的测试工程师，一步一个脚印地走着自己的测试之路，我们可以认为他是相对成功的典范。现实中我们很多测试工程师不是抱怨工资低，就是抱怨自己公司的测试环境不好。其实要想在测试领域走向成功，重要的秘诀就是踏踏实实地学习，认认真真地做好工作。

**案例 2：执著的测试工程师**

B 做测试工作快 6 年了。刚开始的时候，是公司的第一个测试员，虽然公司也在做 ISO9000，但是什么规范都需要自己摸索。虽然在公司里不受重视，仍然坚持下来了，而且大有收获。

但是随着测试工作的不断深入，B 对公司的主流业务从外行变成了内行。而且还发现了一些产品的设计方面的欠缺，在老板和开发主管面前树立了自己的一些威信。至少在一些项目进行需求分析的时候，会来征求 B 的意见。而且，目前做到了不经测试的产品不给客户。

实际上，做任何一行工作，都会遇到不公平。但为什么要去跟别人比呢，只要自己有提高，就是好的。

> 点评：“敢坐冷板凳的人”才是勇敢的人，这位发帖子的测试工程师不但有勇气坐了冷板凳，而且能够坚持下来，直到最后取得了很多个人成绩。“实际上，做任何一行工作，都会遇到不公平。但为什么要去跟别人比呢，只要自己有提高，就是好的。”几句朴实无华的话说出了如何做好测试工作的真谛。

**案例 3：好学而有信心的新手**

C 在一家外企担任兼职测试师的工作，之前从未接触过测试。开始的时候只做一些 Manual test，后来就开始做 Automated test，修改原有的 test cases，或者重写一些 test cases。然后 test 小组的 leader 建议 C 写测试文档，说写文档有利于一个 tester 技术水平的提高。因为必须要熟悉软件项目的整体框架，洞悉软件深层的结构才能写出高质量的测试文档。

于是，C 在网上查了一些关于测试方面的资料，发现测试真的很重要。对一个软件项目而言，老外对软件测试尤其重视。个人认为，国内的软件企业对测试的重视程度还不够，但是毋庸置疑，测试是软件企业产品线上和开发同等重要的。

> 点评：可以看出这是一位很有远见的测试工程师。现实中很多测试工程师是由于不能从事其他工作才从事测试的，因而工作中也是不断地抱怨待遇、团队环境等不能满足自己的要求。在此建议测试工程师，如果选择了这个行业，就应该认真地对待工作，抱怨永远解决不了问题。只有像这位测试工程师一样认真分析自己的行业，才可以有更好的职业化发展，否则还不如换一个自己喜欢的工作内容去试一试。

### 11.3.2 测试一个软件最首要的任务

测试一个软件最首要也是最重要的任务是测试其产品功能说明书。

**1. 概念**

产品功能说明书：对产品最终需要实现的功能的描述。这些功能是最终确定的需要满足的客户需求，也包括是一些软件必须具备的能力。在规范的软件生成的流程中，产品功能说明书应在用户需求评审会议召开后，进行系统的概要设计前确定。

**2. 原因**

（1）很多软件的缺陷都是因为产品功能说明书不够全面，经常更改造成的。

（2）只有详细地阅读了产品功能说明书，确认产品需要实现的功能，才能拟定切实可行的测试方案。

**3. 方法**

（1）参照需求说明书，检查产品功能说明书描述的产品将要实现的功能是否已经完整、准确、一致、合理地描述了产品的功能，并确保这些功能是可测试的。

（2）研究产品说明书是否符合现有的软件设计开发的标准或规范。

（3）研究同类软件，预测产品的最终结果。

如果测试人员发现产品说明书不符合以上几点，该怎么做？

测试人员需要明白，我们的责任是反映产品的缺陷，我们不需要也不能修正产品，所以同发现软件的其他缺陷一样，在发现产品说明书的缺陷后，应该把它们如实并详细地记录下来，呈报给此软件的最终负责人，对并此缺陷的处理情况进行跟踪。注意同发现的软件其他缺陷一样，缺陷列表应该呈报给软件的最终负责人，而不是给相关技术人员或技术主管，因为技术人员可能会以在技术的实现上有难度为推托，拒绝对缺陷的修改。

**4. 目前的可执行度**

（1）很多软件在开发前并没有书面形式的产品说明书。

目前我国的许多软件公司都是小型的公司，在程序开发前都没有一个正式的、完整的、确定的产品说明书，即便是这种情况，产品说明书也是存在的，它存在于软件设计人员、项目负责人的脑海里，在他们心中都有一个软件的轮廓，假定了软件将要实现的功能。我们的测试人员可以在同他们的交流和不断的询问中获得这个非正式的产品说明书，值得注意的是在我们得到这些信息后还需要以书面的形式把它们一一列举出来，再向相关人员请教，最后做到能完整、准确、一致、合理地描述了产品的功能。

（2）测试人员一般不会在项目初期就参与项目。

当前还存在着这样一种问题，公司一般不会让软件测试人员在项目的初期就参与项目，一般要等到软件的雏形出来后才会让软件测试人员着手进行测试。对这种情况，测试人员可以通过已经建立的软件的雏形，揣摩产品说明书，也同上段所说一样，向相关

人员请教，拟定一份书面的完整的、准确的、一致的、合理的产品说明书。值得注意的是，测试人员在运行软件的雏形时，往往会发现一些软件缺陷，这时千万不要局限在这些缺陷上耗费精力，以致忘了拟定产品说明书的主要任务，一定要记住：测试一个软件最首要也是最重要的是测试其产品说明书，在产品说明书明确后，再制定具体的测试案例。

以上两点是指在公司体系不太正规的情况下给测试员的建议，但要能更好地保证软件的质量，或许规范生成软件的整个运作流程更为有效。产品说明书由项目负责人来主持定版比较好，因为他把握着产品发展的方向；在产品说明书定版时的会议应请负责测试的人参加，使他们对产品有一个宏观的认识，但是我们也不赞成测试人员过早的局限于某一项目，如果那样他们会觉得无所事事。

## 11.3.3　测试行业职场小规则

**1．测试人员首先要学会尊重自己**

软件测试人员首先应该尊重自己的劳动成果——软件缺陷报告。很多测试人员都不能清晰地描述一个软件缺陷，尤其分不清缺陷跟踪系统中 Summary 和 Description 的区别，软件缺陷描述——Summary 和 Description 中就输入了完全一样的内容。

严格的讲，Summary 通常用于概要性地描述软件缺陷内容或者发生问题时的现象，主要用于项目经理进行缺陷分配，因此要用最简短、精悍的语言来描述是什么缺陷，使项目经理很快明白是什么问题、应该分配给哪个开发人员；而 Description 则用来描述缺陷的详细信息，通常描述缺陷的重现步骤，主要供开发人员修改缺陷时候查看。

软件缺陷报告是测试人员最直接的劳动成果，因此应该认真地描述自己所提交的每一个软件缺陷，这也是尊重自己劳动成果的一种表现。缺陷描述不清晰，不但将会增加沟通成本，更重要的是不会得到开发人员的认可与尊重。测试人员在为开发人员的成果——产品找问题的同时，也要保证自己的成果没有问题。

因此，作为测试人员首先要学会清晰、准确地报告一个缺陷，这将是与开发人员互相赢得对方尊重的开端，也是尊重自己的表现。试想，如果自己都不爱惜自己的劳动成果，那别人如何会尊重你的成果呢？

**2．不要只做领导告诉你的事，而是做需要做的事**

在一家公司，如果你只会做领导要你做的事，你会没有一点突破，而且你的领导会觉得你没有一点思考或创新能力。

所以不只是按领导分配给你做的事，你还要帮领导想，做这件事时你可以做得更好的方面，当然最好是你的领导没有想到的。

**3．为你的人生只做加法，不做减法**

什么是加法，就是得到领导或同事认可，反之减法你也知道了吧。

可能你无法完全做到，但你可以避免，做到风险控制到最低点。

比如说你发现一个 Bug，你自己无法确认，这时你应该怎么做呢？

直接汇报给上级吗？可千万别这么做，你可以找身边的同事或资深人士沟通一下，做到心中有数，最好有完全的解决方案然后才汇报。

**4. 先做人后做事**

在一个新的环境里，你学会了先做人后做事了吗？

也许你很奇怪，应该尽快出业绩让大家看到我吗？有时你觉得业绩重要吗？还是你在工作中犯一个错误面临被裁的危险重要呢？其实做人和做事一样重要，因为新的环境对你来说是陌生的，一切都要重新开始，就算你很聪明，学习或熟悉也总有一个过程吧。至少在第 1 个星期里，你可以先学会做人。

**5. 如何弥补自己的短处**

相信没有人是万能的，比如说英语水平，比如从事软件外包工作，经常与国外的同事沟通。所以应该给自己排列出英语的学习计划，在不知不觉中你会发现自己会有很大的进步。这也是你必须做的事情。

不要在同一时间段内给自己定 N 个目标，即使你有 N 个短处，你也可以分时间段一个一个去增高，让整个水桶的水慢慢地增加。

**6. 如何与开发人员保持友好关系**

测试工程师刚开始进入测试的角色时，一发现 Bug 都非常兴奋。都会大摇大摆去告诉开发人员，很得意地说这是你的问题，结果应该是怎么样。

一段时间后发现开发都会怨气大增，平时打招呼也不像之前那么热情了。

后来逐渐意识到这点，及时端正自己的态度，现在和开发都是称兄道弟。

这里有一些小小的潜规则要给新的测试朋友：

① 每个 Bug 重复多做几遍，直到你可以熟练的给开发演示。

② 如果自己的英语不好，而又不得不给美国的同事提交 Bug，最好有详细的操作步骤，配上图片说明，让别人看图就能明白意思，当然你还有能力可以外加几句英文说明一下。但忌大面积用文字说明。

③ 不是每个 Bug 都需要提交到 Bug 管理软件，有时 Bug 即使严重程度很高，如果想保持与开发的关系，可以私下里给他。也许这是一条妙招，百用不厌。但你自己要做一个 Bug 表，修正后的版本一定要记得做 Check。可你也不能经常这样做，给个 1～2 回就可以了，千万不能过 3，否则就没有用了。如果真的是严重到让系统崩溃的 Bug，就要加入 Bug 管理软件了。

④ 学会给点小恩小惠，自己平时可以准备一点小零食在办公室。当然你要有心不要只是一般的饼干，而是精心准备的，同样不是经常，只是偶尔。要学会分享给所有的同事，当你和所有的同事一起吃美食的感觉，你会觉得非常的美妙。

**7. 要学会并善于思考**

在工作中为什么要学会思考，做测试不只是简单的执行，你要想想你同事做的测试

用例有没有到位，是不是已经是最好的，如果让你来做是不是可以更简单就可以达到目标。

**8．对自己所从事的工作，要有热情**

在这里想引用一位测试工程师博客中的一段：我不知道你现在做的工作或测试是不是你喜欢的，但我想告诉你，暂时我还没有发现比测试能让我疯狂的工作，也许以后有。因为现在没有，我就很开心地做我的测试，很享受现在的测试工作。有时觉得真的到了一种疯狂的地步。虽然测试技术不是很高，但能让自己很开心，很认真地工作，也觉得值得，关键是测试给了我很多，至少实现自己的价值，有一种满足感。

不要只是把测试工作限在 8 小时之内，也许只做国内的软件测试的朋友可以吧。但我觉得应该在 8 小时之外，如果你的系统在美国时间上线，那你就应该在上线后再做一次完整的测试。如果没有发现大的问题，你才能安心去睡觉。

**9．要学会拉外联**

因为毕竟刚入这行，什么都不懂，也不能只局限在自己的小井里看天。要走出去，寻求专家和能人的帮助。

## 11.4　软件测试认识中的误区

我国软件业主要集中在最终客户软件项目的开发上，通用软件商品的开发比较少，所以我国的软件开发水平和软件业发达国家（如美国、印度、爱尔兰等）相比，有一定的距离，主要体现在软件测试和质量保证的水平。

随着我国加入 WTO 之后，软件测试越来越受到人们的注意。人们认识到，要提高软件质量，软件测试是极其重要的一个环节。但是，相对于软件开发而言，软件测试还不为众人所了解。很多软件开发人员，包括多数软件企业的高层管理人员，由于缺乏软件测试的知识和实践经验，对软件测试的认识还有很多误区，这对软件测试工作极为不利，必须加以澄清，才可能提高软件产品的质量。

误区一：如果发布出去的软件有质量问题，那是软件测试人员的错。

软件测试是一种有效提高软件质量的手段，但即使在投入上有所保证，测试也并不能百分之百地发现所有质量隐患。况且，软件的高质量不是靠测试测出来的，而是软件开发过程中每一个环节都要有质量意识，做好检查、审查等工作，才能保证质量。任何一个环节出问题，都有可能带来质量的隐患。

误区二：软件测试技术要求不高，至少比编程容易多了。

很多人认为软件测试就是运行一下软件，然后看看结果对不对。实际上，软件测试不仅仅只是运行或操作软件，还要涉及测试环境的建立、测试用例的设计等技术问题。当采用白盒测试方法时，需要良好的编程经验；在进行自动化测试时，需要写测试脚本，也需要良好的编程经验。如何在有限的资金投入下，提高软件测试的效率和保证产品的

高质量是一件较困难的事情。所以，一位优秀的软件测试人员不仅要掌握各种测试技术，还要具备丰富的编程经验和对 Bug 的敏感，从一定意义上说，对测试人员的要求比编程人员的要求还要高。

误区三：有时间就多测试一些，来不及就少测试一些。

软件测试并不是可有可无的，测多少、怎样测也不是随心所欲的。规范化的软件开发过程需要对软件测试早做计划，分配必要的时间、人力和财力等资源，并将其作为项目管理的一个重要部分，进行跟踪、控制和协调。

误区四：软件测试是测试人员的事，与开发人员无关。

为了减小相互的影响，一般要求开发和测试相对独立，但这只是分工上的不同。编程和测试是软件项目相辅相成的两个过程，人员之间的交流、协作和配合是提高整体开发效率的重要因素。而且在实际操作中也会有一些测试，比如单元测试，会由测试人员提供测试用例，由开发人员运行，或全部由编程人员完成。

误区五：根据软件开发瀑布模型，软件测试是开发后期的一个阶段。

不少人根据软件开发瀑布模型，容易得出这样一种结论："程序代码写完之后再进行测试"，这是错误的，这种错误的观念会给测试工作带来很多问题。

生命周期中的"测试阶段"表明在该阶段主要工作是测试，即到了"测试阶段"，测试的主要任务是执行测试、运行测试脚本、测试结果分析和递交测试报告，而测试的大量准备工作，诸如测试计划、测试用例设计以及测试脚本编写等，实际是在还没有开始写程序之前就开始了，可以说项目什么时候开始，测试工作也就什么时候开始。如果到"测试阶段"才开始启动测试工作，那为时已晚，没有计划，没有测试用例，测试也就很容易是"走过场"。

即使程序测试，也不是要等到软件程序全部完成后才开始运行。针对每个程序单元、模块，可以进行单元测试；把程序单元或模块进行集成时，需要进行相应的集成测试；在进行集成测试和系统测试之前，就要准备测试环境。所以，编程与测试几乎是同步进行的。

软件产品中的质量问题发现越晚，修正错误的代价就越大。就这一点而言，也要求软件测试尽早开始，包括需求分析和设计文档的审查。

在实践中，除对软件测试不够重视之外，还可能存在一些对软件测试本身不正确的看法。

- 认为测试工作不如设计和编码那样容易取得进展，难以给测试人员带来某种成就感。
- 以发现软件错误为目标的测试是非建设性的，甚至是破坏性的，测试中发现错误是对责任人工作的一种否定。
- 测试工作枯燥无味，不能引起人们的兴趣。
- 测试工作是艰苦而细致的工作。
- 对自己编写的程序盲目自信，在发现错误后，顾虑别人对自己开发能力的看法。

# 附录　软件测试专业术语对照表

## 前　　言

此术语表为国际软件测试认证委员会（ISTQB）发布的标准术语表。此表历经数次修改、完善，集纳了计算机行业界、商业界及政府相关机构的见解及意见，在国际化的层面上达到了罕有的统一性及一致性。参与编制此表的国际团体包括澳大利亚、比利时、芬兰、德国、印度、以色列、荷兰、挪威、葡萄牙、瑞典、英国和美国。

多数软件测试工程师使用 1998 年发布的 BS 7925-1 标准。英国信息系统考试委员会（ISEB）也以此标准作为基础级别和从业级别认证的首要参考标准。BS 7925-1 标准最初是围绕着单元测试撰写的，自发布之后许多旨在改进和扩展此标准，以覆盖更广义范围的软件测试领域的新概念和提议不断涌现。最新版的 BS7925-1 标准中的软件测试词汇吸纳、融合了上述概念和提议。此国际软件测试认证委员员（ISTQB）发布的标准术语表即是以最新版的 BS 7925-1 标准为基础制定的国际化软件测试标准术语。

**1．简介**

行业界、商业界、政府及学术机构曾经花费大量精力和时间以解释和区分一些常见的软件测试专业术语以期在各社会部门或机构之间达成交流，例如，语句覆盖（Statement Coverage）和条件覆盖（Decision Overage）；测试套件（Test Suite）、测试规格说明书（Test Specification）和测试计划（Test Plan）等。上述机构与专职机构定义的同名术语在含义上又往往有很大偏差。

**2．范畴**

本文档旨在提供概念、条款和定义为软件测试及相关从业人员进行有效交流的平台。

**3．结构**

术语表中的词汇按字母顺序排列。术语如有同义词汇，本术语表解释最通用的词汇，其同义词汇会的仅被列出，不予重复解释。例如结构测试（Structural Testing）和白盒测试（White Box Testing）。此类同义词在术语表中用“参见”列出，以便读者检索。“参见”往往连接着广义和狭义词或含义重叠的词汇。

**4．标准参考**

至截稿日期，此标准有效版本为 1.2。如所有其他标准一样，本术语表仍需根据以下相关标准的最新版本不断修正。此标准由 IEC 和 ISO 成员根据目前有效的国际相关

标准进行更新。

-BS 7925-2:1998. Software Component Testing.

-DO-178B:1992. Software Considerations in Airborne Systems and Equipment Certification, Requirements and Technical Concepts for Aviation (RTCA SC167).

-IEEE 610.12:1990. Standard Glossary of Software Engineering Terminology.

-IEEE 829:1998. Standard for Software Test Documentation.

-IEEE 1008:1993. Standard for Software Unit Testing.

-IEEE 1012:1986. Standard for Verification and Validation Plans.

-IEEE 1028:1997. Standard for Software Reviews and Audits.

-IEEE 1044:1993. Standard Classification for Software Anomalies.

-IEEE 1219:1998. Software Maintenance.

-ISO/IEC 2382-1:1993. Data processing-Vocabulary-Part 1: Fundamental terms.

-ISO 9000:2000. Quality Management Systems – Fundamentals and Vocabulary.

-ISO/IEC 9126-1:2001. Software Engineering – Software Product Quality – Part 1: Quality Characteristis and Sub-characteristics.

-ISO/IEC 12207:1995. Information Technology – Software Life Cycle Processes.

-ISO/IEC 14598-1:1996. Information Technology – Software Product Evaluation-Part 1: General Overview.

| A | | |
|---|---|---|
| abstract test case | 抽象测试用例 | 参见 high level test case |
| acceptance | 验收 | 参见 acceptance testing |
| acceptance criteria | 验收准则 | 为了满足组件或系统使用者、客户或其他授权实体的需要，组件或系统必须达到的准则。[IEEE610] |
| acceptance testing | 验收测试 | 一般由用户/客户进行的确认是否可以接受一个系统的验证性测试。是根据用户需求，业务流程进行的正式测试以确保系统符合所有验收准则。[与 IEEE 610 一致] |
| accessibility testing | 可达性测试 | 可达性测试就是测试残疾人或不方便的人们使用软件或者组件的容易程度[Gerrard]。即被测试的软件是否能够被残疾或者部分有障碍人士正常使用，这其中也包含了正常人在某些时候发生暂时性障碍的情况下正常使用，如怀抱婴儿等 |
| accuracy | 准确性 | 软件产品的提供的结果的正确性、一致性和精确程度的能力。[ISO9126] 参见 functionality testing |
| actual outcome | 实际结果 | 参见 actual result |
| actual result | 实际结果 | 组件或系统测试之后产生或观察到的行为 |
| ad hoc review | 临时评审 | 非正式评审（和正式的评审相比） |

续表

| | | |
|---|---|---|
| A | | |
| ad hoc testing | 随机测试 | 非正式的测试执行。即没有正式的测试准备、规格设计和技术应用，也没有期望结果和必须遵循的测试执行指南 |
| adaptability | 适应性 | 软件产品无须进行额外修改，而适应不同特定环境的能力。[ISO9126] 参见 protability |
| agile testing | 敏捷测试 | 对使用敏捷方法，如极限编程（extreme programming）开发的项目进行的软件测试，强调测试优先行的设计模式，见 test driven development |
| algorithm test [TMap] | 算法测试 | 参见 branch testing |
| alpha testing | Alpha 测试 | 由潜在用户或者独立的测试团队在开发环境下或者模拟实际操作环境下进行的测试，通常在开发组织之外进行。通常是对现货软件（COTS）进行内部验收测试的一种方式 |
| analyzability | 可分析性 | 软件产品缺陷或运行失败原因可被诊断的能力，或对修改部分的可识别能力。[ISO9126] 参见 maintainability |
| analyzer | 分析器 | 参见 static analyzer |
| anomaly | 异常 | 任何基于需求文档、设计文档、用户文档、标准或者个人的期望和预期之间偏差的情况，都可以称为异常。异常可以在但不限于下面的过程中识别：评审(review)、测试分析(test analysis)、编译(compilation)、软件产品或应用文档的使用等。参见 defect, deviation, error, fault, failure, incident, problem |
| arc testing | 弧测试 | 参见 branch testing |
| attractiveness | 吸引力 | 软件产品吸引用户的能力。[ISO9126]参见 usability |
| audit | 审计 | 对软件产品或过程进行的独立评审，来确认产品是否满足标准、指南、规格说明书以及基于客观准则的步骤等，包括下面的文档：（1）产品的内容与形式；（2）产品开发应该遵循的流程；（3）度量符合标准或指南的准则。[IEEE1028] |
| audit trail | 审计跟踪 | 以过程输出作为起点，追溯到原始输入（如数据）的路径。有利于缺陷分析和过程审计的开展。[与 TMap 一致] |
| automated testware | 自动测试件 | 用于自动化测试中的测试件，如工具脚本 |
| availability | 可用性 | 用户使用系统或组件的可操作和易用的程度，通常以百分比的形式出现。[IEEE610] |

续表

| B | | |
|---|---|---|
| back-to-back testing | 比对测试 | 用相同的输入，执行组件或系统的两个或多个变量，在产生偏差的时候，对输出结果进行比较和分析 |
| baseline | 基线 | 通过正式评审或批准的规格或软件产品。以它作为继续开发的基准。并且在变更的时候，必须通过正式的变更流程来进行。[与 IEEE610 一致] |
| basic block | 基本块 | 一个或多个连续可执行的语句块，不包含任何分支语句 |
| basis test set | 基本测试集 | 根据组件的内部结构或规格说明书设计的一组测试用例集。通过执行这组测试用例可以保证达到 100%的指定覆盖准则（coverage criterion）的要求 |
| bebugging | 错误散播 | 参见 error seeding |
| behavior | 行为 | 组件或系统对输入值和预置条件的反应 |
| benchmark test | 基准测试 | （1）为使系统或组件能够进行度量和比较而制定的一种测试标准；（2）用于组件或系统之间进行的比较或和（1）中提到的标准进行比较的测试。[与 IEEE610 一致] |
| bespoke software | 定制软件 | 为特定的用户定制开发的软件。与之对比的是现货软件（off-the-shelf soft ware） |
| best practice | 最佳实践 | 在界定范围内，帮助提高组织能力的有效方法或创新实践，通常被同行业组织视为最佳的方法或实践 |
| beta testing | Beta 测试 | 用户在开发组织外，没有开发人员参与的情况下进行的测试，检验软件是否满足客户及业务需求。这种测试是软件产品获得市场反馈进行验收测试的一种形式 |
| big-bang testing | 大爆炸测试 | 非增量集成测试的一种方法，测试的时候将软件单元、硬件单元或者两者同时进行，而不是阶段性的，集成到组件或者整个系统中去进行测试。[与 IEEE610 一致]参见 integration testing |
| black-box technique | 黑盒技术 | 参见 black-box test design technique |
| black-box testing | 黑盒测试 | 不考虑组件或系统内部结构的功能或非功能测试 |
| black-box test design technique | 黑盒测试设计技术 | 基于系统功能或非功能规格说明书来设计或者选择测试用例的技术，不涉及软件内部结构 |
| bottom-up testing | 自底向上测试 | 渐增式集成测试的一种，其策略是先测试底层的组件，以此为基础逐步进行更高层次的组件测试，直到系统集成所有的组件。参见 integration testing |
| boundary value | 边界值 | 通过分析输入或输出变量的边界或等价划分（equivalence partition）的边界来设计测试用例，如取变量的最大值、最小值、中间值、比最大值大的值、比最小值小的值等 |

续表

| | | |
|---|---|---|
| B | | |
| boundary value analysis | 边界值分析 | 一种黑盒设计技术（black-box test design technique），基于边界值进行测试用例的设计 |
| boundary value coverage | 边界值覆盖 | 执行一个测试套件（test suite）所能覆盖的边界值（boundary value）的百分比 |
| boundary value testing | 边界值测试 | 参见 boundary value analysis |
| branch | 分支 | 在组件中，控制从任何语句到其他任何非直接后续语句的一个条件转换，或者是一个无条件转换。例如，case, jump, go to, if-then -else 语句 |
| branch condition | 分支条件 | 参见条件（condition） |
| branch condition combination coverage | 分支条件组合覆盖 | 参见 multiple condition coverage |
| branch condition combination testing | 分支条件组合测试 | 参见 multiple condition testing |
| branch condition coverage | 分支条件覆盖 | 参见 condition coverage |
| branch coverage | 分支覆盖 | 执行一个测试套件(test suite)所能覆盖的分支(branch)的百分比。100%的分支覆盖（branch coverage）是指100%判定条件覆盖（decision covergate）和 100%的语句覆盖（statement covergage） |
| bug | 缺陷 | 参见 defect |
| bug report | 缺陷报告 | 参见 defect report |
| business process-based testing | 基于业务过程测试 | 一种基于业务描述和/或业务流程的测试用例设计方法 |
| C | | |
| Capability Maturity Model（CMM） | 能力成熟度模型 | 描述有效的软件开发过程关键元素的一个五个等级的框架，能力成熟度模型包含了在软件开发和维护中计划、工程和管理方面的最佳实践（best practice），缩写为 CMM。[CMM] |
| Capability Maturity Model Integration（CMMI） | 能力成熟度模型集成 | 描述有效的软件产品开发和维护过程的关键元素框架，能力成熟度模型集成包含了软件开发计划、工程和管理等方面的最佳实践，是 CMM 的指定的继承版本 |
| capture/playback tool | 捕获/回放工具 | 一种执行测试工具，能够捕获在手工测试过程中的输入，并且生成可执行的自动化脚本用于后续阶段的测试（回放过程）。这类工具通常使用在自动化回归测试（regression test）中 |
| capture/replay tool | 捕获/回放工具 | 参见 capture/playback tool |

续表

| C | | |
|---|---|---|
| CASE | 计算机辅助软件工程 | Computer Aided Software Engineering 的首字母缩写 |
| CAST | 计算机辅助软件测试 | Computer Aided Software Testing 的首字母缩写，参见 test automation。在测试过程中使用计算机软件工具进行辅助的测试 |
| cause-effect graph | 因果图 | 用来表示输入（原因）与结果之间关系的图表，因果图可以用来设计测试用例 |
| cause-effect graphing | 因果图技术 | 通过因果图（case-effect graph）设计测试用例的一种黑盒测试设计技术 |
| cause-effect analysis | 因果分析 | 参见因果图技术（case-effect graphing） |
| cause-effect decision table | 因果决策表 | 参见决策表（decision table） |
| certification | 认证 | 确认一个组件、系统或个人具备某些特定要求的过程，比如通过了某个考试 |
| changeability | 可变性 | 软件产品适应修改的能力，[ISO9126] 参见 maintainability |
| change control | 变更控制 | 参见 configuration control |
| change control board | 变更控制委员会 CCB | 参见 configuration control board |
| checker | 检验员 | 参见评审员（reviewer） |
| Chow's coverage metrics | N 切换覆盖度量 | 参见 N 切换覆盖（N-switch coverage）[Chow] |
| classification tree method | 分类树方法 | 运用分类树法而进行的一种黑盒测试设计技术，通过输入和/或输出域的组合来设计测试用例[Grochtmann] |
| code | 代码 | 计算机指令和数据定义在程序语言中的表达形式或是汇编程序、编译器或其他翻译器的一种输出形式 |
| code analyzer | 代码分析器 | 参见静态分析器（static code analyzer） |
| code coverage | 代码覆盖 | 一种分析方法，用于确定软件的哪些部分被测试套件（test suite）覆盖到了，哪些部分没有。如语句覆盖（statement covergage）、判定覆盖（decision coverage）和条件覆盖（condition covergate） |
| code-based testing | 基于代码的测试 | 参见 white-box testing |
| co-existence | 共存性 | 软件产品与通用环境下与之共享资源的其他独立软件之间共存的能力。[ISO9126] 参见可移植性（portability） |
| commercial off-the-shelf software | 商业现货软件 | 参见现货软件（off-the shelf software） |
| comparator | 比较器 | 参见 test comparator |

续表

| C | | |
|---|---|---|
| compiler | 编译器 | 将高级命令语言编写的程序翻译成能运行的机器语言的工具[IEEE610] |
| complete testing | 完全测试 | 参见穷尽测试（exhaustive testing） |
| completion criteria | 完成准则 | 参见退出准则（exit criteria） |
| complexity | 复杂性 | 系统或组件的设计和/或内部结构难以理解、维护或验证的程度。参见 cyclomatic complexity |
| compliance | 一致性 | 软件产品与法律和类似规定的标准、惯例或规则的一致性方面的能力。[ ISO9126] |
| compliance testing | 一致性测试 | 确定组件或系统是否满足标准的测试过程 |
| component | 组件 | 一个可被独立测试的最小软件单元 |
| component integration testing | 组件集成测试 | 为发现集成组件接口之间和集成组件交互产生的缺陷而执行的测试 |
| component specification | 组件规格说明 | 根据组件的功能定义为特定输入而应该产生的输出规格进行的功能性和非功能性行为的描述。例如：资源使用（resource utilization） |
| compound condition | 复合条件 | 通过逻辑操作符（AND, OR 或者 XOR）将两个或多个简单条件连接起来：如，“A>0 AND B<1000” |
| concrete test case | 具体测试用例 | 参见低阶测试用例（low level test case） |
| concurrency testing | 并发测试 | 测试组件或系统的两个或多个活动在同样的间隔时间内如何交叉或同步并发。[与 IEEE610 一致] |
| condition | 条件 | 一个可被判定为真、假（true, false）的逻辑表达式，如 A>B |
| condition combination coverage | 条件组合覆盖 | 参见多条件覆盖（multiple condition coverage） |
| condition combination testing | 条件组合测试 | 参见多条件测试（multiple condition testing） |
| condition coverage | 条件覆盖 | 执行测试套件（test suite）能够覆盖到的条件百分比。100%的条件覆盖要求测试到每一个条件语句真、假（true, false）的条件 |
| condition determination coverage | 条件决定覆盖 | 执行测试套件（test suite）覆盖到的能够独立影响判定结果的单个条件的百分比。100%的条件决定覆盖意味着 100%的判定条件覆盖 |
| condition determination testing | 条件决定测试 | 一种白盒测试技术，是对能够独立影响决策结果的单独条件的测试 |
| condition testing | 条件测试 | 一种白盒测试技术，设计测试用例以执行条件的结果 |
| condition outcome | 条件结果 | 条件判定的结果，为真或假 |

续表

| C | | |
|---|---|---|
| confidence test | 置信测试 | 参见冒烟测试（smoke testing） |
| configuration | 配置 | 根据定义的数值、特性及其相关性综合设置一个组件或者系统 |
| configuration auditing | 配置审核 | 对配置库及配置项的内容进行检查的过程，比如检查标准的一致性。 [IEEE610] |
| configuration control | 配置控制 | 配置管理的一个方面，包括在正式配置完成之后对配置项进行评价、协调、批准或撤销，以及变更修改的控制。 [IEEE610] |
| configuration control board（CCB） | 配置控制委员会 | 负责评估、批准或拒绝配置项修改的组织，此组织应确保被批准的配置修改的执行。 [IEEE610] |
| configuration identification | 配置标识 | 配置管理的要素之一，包括选择配置项，并在技术文档中记录其功能和物理特性。[IEEE610] |
| configuration item | 配置项 | 配置管理中的硬件、软件或软、硬件结合体的集合，在配置管理过程中通常被当做一个实体。[IEEE610] |
| configuration management | 配置管理 | 一套技术和管理方面的监督原则，用于确定和记录一个配置项的功能和物理属性、控制对这些属性的变更、记录和报告变更处理和实现的状态，以及验证与指定需求的一致性。[IEEE610] |
| configuration management tool | 配置管理工具 | 支持对配置项进行识别、控制、变更管理、版本控制和发布配置项基线（baseline）的工具。[IEEE610] |
| configuration testing | 配置测试 | 参见可移植性测试（portability testing） |
| confirmation testing | 确认测试 | 参见再测试（re-testing） |
| conformance testing | 一致性测试 | 参见符合性测试（compliance testing） |
| consistency | 一致性 | 在系统或组件的各组成部分之间和文档之间无矛盾，一致，符合标准的程度。[IEEE610] |
| control flow | 控制流 | 执行组件或系统中的一系列顺序发生的事件或路径 |
| control flow graph | 控制流图 | 通过图形来表示组件或系统中的一系列顺序发生的事件或路径 |
| control flow path | 控制流路径 | 参见路径（path） |
| conversion testing | 转换（移植） 测试 | 用于测试已有系统的数据是否能够转换到替代系统上的一种测试 |
| COTS | 现货软件 | commercial off-the-shelf software 的首字母缩写。参见 off-the-shelf software |
| coverage | 覆盖 | 用于确定执行测试套件所能覆盖项目的程度，通常用百分比来表示 |

续表

| | | |
|---|---|---|
| **C** | | |
| coverage analysis | 覆盖分析 | 对测试执行结果进行特定的覆盖项分析，判断其是否满足预先定义的标准，是否需要设计额外的测试用例 |
| coverage item | 覆盖项 | 作为测试覆盖的基础的一个实体或属性，如等价划分（equivalent partitions）或代码语句（code statement）等 |
| coverage tool | 覆盖工具 | 对执行测试套件（test suite）能够覆盖的结构元素如语句（statement）、分支（branch）等进行客观测量的工具 |
| custom software | 定制软件 | 参见 bespoke software |
| cyclomatic complexity | 圈复杂度 | 程序中独立路径的数量。一种代码复杂度的衡量标准，用来衡量一个模块判定结构的复杂程度，数量上表现为独立现行路径条数，即合理的预防错误所需测试的最少路径条数，圈复杂度大说明程序代码可能质量低且难于测试和维护，根据经验，程序的可能错误和高的圈复杂度有着很大关系。圈复杂度=L–N＋2P，其中 L 表示为结构图（程序图）的边数；N 为结构图（程序图）的节点数目；P 为无链接部分的数目。[与 McCabe 一致] |
| cyclomatic number | 圈数 | 参见 cyclomatic complexity |
| **D** | | |
| daily build | 每日构建 | 每天对整个系统进行编译和链接的开发活动，从而保证在任何时候包含所有变更的完整系统是可用的 |
| data definition | 数据定义 | 给变量赋了值的可执行语句 |
| data driven testing | 数据驱动测试 | 将测试输入和期望输出保存在表格中的一种脚本技术。通过这种技术，运行单个控制脚本就可以执行表格中所有的测试。像录制/回放这样的测试执行工具经常会应用数据驱动测试方法。[Fewster and Graham]，参见 keyword driven testing |
| data flow | 数据流 | 数据对象的顺序的和可能的状态变换的抽象表示，对象的状态可以是创建、使用和销毁。[Beizer] |
| data flow analysis | 数据流分析 | 一种基于变量定义和使用的静态分析（static analysis）模式 |
| data flow coverage | 数据流覆盖 | 执行测试套件（test suite）能够覆盖已经定义数据流的百分比 |
| data flow testing | 数据流测试 | 一种白盒测试设计技术，设计的测试用例用来测试变量的定义和使用路径 |
| data integrity testing | 数据完整性测试 | 参见 database integrity testing |

续表

| D | | |
|---|---|---|
| database integrity testing | 数据库完整性测试 | 对数据库的存取和管理进行测试的方法和过程，确保数据库如预期一样进行存取、处理等数据功能，同时也确保数据在存取过程中没有出现不可预料的删除、更新和创建 |
| dead code | 死代码 | 参见 unreachable code |
| debugger | 调试器 | 参见 debugging tool |
| debugging | 调试 | 发现、分析和去除软件失败根源的过程 |
| debugging tool | 调试工具 | 程序员用来复现软件失败、研究程序状态并查找相应缺陷的工具。调试器可以让程序员单步执行程序、在任何程序语句中终止程序和设置、检查程序变量 |
| decision | 判定 | 有两个或多个可替换路径控制流的一个程序控制点。也是连接两个或多个分支的节点 |
| decision condition coverage | 判定条件覆盖 | 执行测试用例套件（test suite） 能够覆盖的条件结果（condition outcomes）和判定结果（decision outcomes）的百分比，100% 的判定条件覆盖意味着 100%的判定覆盖和 100% 的条件覆盖 |
| decision condition testing | 判定条件测试 | 一种白盒测试（white-box）设计技术，设计的测试用例用来测试条件结果（condition outcoems）和判定结果（decision outcomes） |
| decision coverage | 判定覆盖 | 执行测试套件能够覆盖的判定结果（decsion outcomes）的百分比。100%的判定覆盖（decision converage）意味着 100 的分支覆盖（branch coverage）和 100%的语句覆盖（statement coverage） |
| decision table | 决策表 | 一个可用来设计测试用例的表格，一般由条件桩、行动桩和条件规则条目和行动规则条目组成 |
| decision table testing | 决策表测试 | 一种黑盒测试设计技术，设计的测试用例用来测试判定表中各种条件的组合 |
| decision testing | 决策测试 | 白盒测试设计技术的一种，设计测试用例来执行判定结果 |
| decision outcome | 判定结果 | 判定的结果（可以来决定执行哪条分支） |
| defect | 缺陷 | 可能会导致软件组件或系统无法执行其定义的功能的瑕疵，如错误的语句或变量定义。如果在组件或系统运行中遇到缺陷，可能会导致运行的失败 |
| defect density | 缺陷密度 | 将软件组件或系统的缺陷数和软件或者组件规模相比的一种度量（标准的度量术语包括，如每千行代码、每个类或功能点存在的缺陷数） |
| Defect Detection Percentage（DDP） | 缺陷发现百分比 | 在一个测试阶段发现的缺陷数除以在测试阶段和之后其他阶段发现的缺陷总数所得的百分比数 |

续表

| D | | |
|---|---|---|
| defect management | 缺陷管理 | 发现、研究、处置、去除缺陷的过程。包括记录缺陷、分类缺陷和识别缺陷可能造成的影响。[与 IEEE1044 一致] |
| defect management tool | 缺陷管理工具 | 一个方便记录和跟踪缺陷的工具，通常包括以缺陷修复操作流程为引导的任务分配、缺陷修复、重新测试等行为的跟踪和控制，并且提供文档形式的报告。参见 incident management tool |
| defect masking | 缺陷屏蔽 | 一个缺陷阻碍另一个缺陷被发现的情况[与 IEEE610 一致] |
| defect report | 缺陷报告 | 对造成软件组件或系统不能实现预期功能的缺陷进行描述的报告文件 |
| defect tracking tool | 缺陷跟踪工具 | 参见 defect management tool |
| definition-use pair | 定义—使用对 | 变量在程序中定义和使用的相关性，变量使用包括变量计算（如乘）或者变量引导程序执行一条路径（预定义） |
| deliverable | 交付物 | 过程中生成的交付给客户的（工作）产品 |
| design-based testing | 基于设计的测试 | 根据组件或系统的构架或详细设计设计测试用例的一种测试方法（如组件或系统之间接口的测试） |
| desk checking | 桌面检查 | 通过手工模拟执行来对软件或规格说明而进行的测试。参见 static analysis |
| development testing | 开发测试 | 通常在开发环境下，开发人员在组件或系统实现过程中进行的正式或非正式的测试。[与 IEEE610 一致] |
| deviation | 偏离 | 参见 incident |
| deviation report | 偏离报告 | 参见 incident report |
| dirty testing | 负面测试 | 参见 negative testing |
| documentation testing | 文档测试 | 关于文档质量的测试，如对用户手册或安装手册的测试 |
| domain | 域 | 一个可供有效输入和/或输出值选择的集合 |
| driver | 驱动器 | 代替某个软件组件来模拟控制和/或调用其他组件或系统的软件或测试工具。[与 TMap 一致] |
| dynamic analysis | 动态分析 | 组件或系统的执行过程中对其行为评估的过程，例如对内存性能、CPU 使用率等的估算。[与 IEEE610 一致] |
| dynamic analysis tool | 动态分析工具 | 为程序代码提供实时信息的工具。通常用于识别未定义的指针，检测指针算法和内存地址分配、使用及释放的情况以及对内存泄露进行标记 |
| dynamic comparison | 动态比较 | 在软件运行过程中（例如用测试工具执行），对实际结果和期望结果的比较 |
| dynamic testing | 动态测试 | 通过运行软件的组件或系统来测试软件 |

续表

| E | | |
|---|---|---|
| efficiency | 效率 | 一定条件下根据资源的使用情况，软件产品能够提供适当性能的能力。[ISO 9126] |
| efficiency testing | 效率测试 | 确定测试软件产品效率的测试过程 |
| elementary comparison testing | 基本比较测试 | 一种黑盒测试设计技术，根据判定条件覆盖的理念，设计测试用例来测试软件各种输入的组合。[TMap] |
| emulator | 仿真器 | 一个接受同样输入并产生同样输出的设备、计算机程序或系统。[IEEE610]参见 simulator |
| entry criteria | 入口准则 | 进入下个任务（如测试阶段）必须满足的条件。准入条件的目的是防止执行不能满足准入条件的活动而浪费资源[Gilb and Graham] |
| entry point | 入口点 | 一个组件的第一个可执行语句 |
| equivalence class | 等价类 | 参见 equivalence partition |
| equivalence partition | 等价类划分 | 根据规格说明，输入域或输出域的一个子域内的任何值都能使组件或系统产生相同的响应结果 |
| equivalence partition coverage | 等价划分覆盖 | 执行测试套件能够覆盖到的等价类的百分比 |
| equivalence partitioning | 等价类划分技术 | 黑盒测试用例设计技术，该技术从组件的等价类中选取典型的点进行测试。原则上每个等价类中至少要选取一个典型的点来设计测试用例 |
| error | 错误 | 人为的产生不正确结果的行为。[与 IEEE610 一致] |
| error guessing | 错误推测 | 根据测试人员以往的经验，猜测在组件或系统中可能出现的缺陷以及错误，并以此为依据来进行特殊的用例设计以暴露这些缺陷 |
| error seeding | 错误散播 | 在组件或系统中有意插入一些已知缺陷（defect） 的过程，目的是为了得到缺陷的探测率和除去率，以及评估系统中遗留缺陷的数量。[IEEE610] |
| error tolerance | 容错 | 组件或系统存在缺陷的情况下保持连续正常工作状态的能力。[与 IEEE610 一致] |
| evaluation | 评估 | 参见 testing |
| exception handling | 异常处理 | 组件或系统对错误输入的行为反应。错误输入包括人为的输入、其他组件或系统的输入以及内部失败引起的输入等 |
| executable statement | 可执行语句 | 语句编译后可以转换为目标代码，同时在程序运行的时候可以按步骤执行并且可以对数据进行相应的操作 |
| exercised | 被执行 | 测试用例运行后被执行的语句、判定和程序的结构元素 |
| exhaustive testing | 穷尽测试 | 测试套件包含了软件输入值和前提条件所有可能组合的测试方法 |

续表

| | | |
|---|---|---|
| E | | |
| exit criteria | 出口准则 | 和利益相关者达成一致的系列通用和专门的条件，来正式的定义一个过程的结束点。出口准则的目的可以防止将没有完成的任务错误地看成任务已经完成。测试中使用的出口准则可以来报告和计划什么时候可以停止测试。[与 Gilb 和 Graham 一致] |
| exit point | 出口点 | 组件中最后一个可执行语句 |
| expected outcome | 预期结果 | 参见 expected result |
| expected result | 预期结果 | 在特定条件下根据规格说明或其他资源说明，组件或系统预测的行为 |
| experienced-based test design technique | 基于经验的测试设计技术 | 根据测试人员的经验、知识和直觉来进行用例设计和/或选择的一种技术 |
| exploratory testing | 探索性测试 | 非正式的测试设计技术。测试人员能动地设计一些测试用例，通过执行这些测试用例和在测试中得到的信息来设计新的更好的测试用例。[和 Bach 一致] |
| F | | |
| fail | 失败 | 假如测试的实际结果与预期结果不一样，我们就认为这个测试的状态为失败 |
| failure | 失效 | 组件/系统与预期的交付、服务或结果存在的偏差。[与 Fenton 一致] |
| failure mode | 失效模式 | 失效在物理上或功能上的表现。例如，系统在失效模式下，可能表现为运行缓慢、输出错误或者执行的彻底中断。[IEEE610] |
| failure mode and effect analysis（FMEA） | 失效模式和影响分析（FMEA） | 一个系统的进行风险识别和标识可能的失效模式的系统方法，用来预防失效的发生 |
| failure rate | 失效率 | 指定类型中单位度量内发生失效的数目。例如，单位时间失效数、单位处理失效数、单位计算机运行失效数。[IEEE610] |
| fault | 故障 | 参见 defect |
| fault density | 故障密度 | 参见 defect density |
| fault detection percentage（FDP） | 故障发现率（FDP） | 参见 Defect Detection Percentage（DDP） |
| fault masking | 故障屏蔽 | 参见 defect masking |
| fault tolerance | 故障容限 | 软件产品存在故障或其指定接口遭到破坏时，继续维持特定性能级别的能力。[ISO9126] 参见 reliability |
| fault tree analysis | 故障树分析 | 分析产生故障原因的一种方法 |
| feasible path | 可达路径 | 可通过一组输入值和入口条件而执行到的一条路径 |

续表

| | | |
|---|---|---|
| F | | |
| feature | 特性 | 需求文档指定的或包含的一个组件或者系统的属性（如 reliability, usability 或者 designconstraints）。[与 IEEE1008 一致] |
| field testing | 现场测试 | 参见 beta testing |
| finite state machine | 有限状态机 | 包含有限数目状态和状态之间转换的一种计算模型，同时可能伴随一些可能的（触发）行为。[IEEE 610] |
| finite state testing | 有限状态测试 | 参见 state transition testing |
| formal review | 正式评审 | 对评审过程及需求文档化的一种特定的评审，如检视（inspection） |
| frozen test basis | 冻结测试基准 | 测试基准文档，只能通过正式的变更控制过程进行修正。参见 baseline |
| functional point analysis（FPA） | 功能点分析 | 对信息系统功能进行规模度量的一种方法。该度量独立于具体的技术实现，可以作为生产率度量、资源需求估算和项目控制的基础 |
| functional integration | 功能集成 | 合并组件/系统以尽早实现基本功能的一种集成方法。参见 integration testing |
| functional requirement | 功能需求 | 指定组件/系统必须实现某项功能的需求。[IEEE610] |
| functional test design technique | 功能测试设计技术 | 通过对组件或系统的功能规格说明分析来进行测试用例的设计和/或选择的过程，该过程不涉及软件的内部结构。参见 black box-test design lechnique |
| functional testing | 功能测试 | 通过对组件/系统功能规格说明的分析而进行的测试。参见 black-box testing |
| functionality | 功能性 | 软件产品在规定条件下使用时，所提供的功能达到宣称的和隐含需求的能力。[ISO9126] |
| functionality testing | 功能性测试 | 判断软件产品功能性的测试过程 |
| G | | |
| glass box testing | 玻璃盒测试 | 参见 white-box testing |
| H | | |
| heuristic evaluation | 启发式评估 | 一种静态可用性测试技术，判断用户接口和公认的可用性原则的符合度。 |
| high level test case | 概要测试用例 | 没有具体的（实现级别）输入数据和预期结果的测试用例。实际值没有定义或是可变的，而用逻辑概念来代替。参见 low level test case |
| horizontal traceability | 水平可追踪性 | 一个测试级别的需求和相应级别的测试文档（如测试计划、测试设计规格、测试用例规格和测试过程规格或测试脚本）之间的可追踪性 |

续表

| I | | |
|---|---|---|
| impact analysis | 影响分析 | 对需求变更所造成的开发文档、测试文档和组件的修改的评估 |
| incident | 事件 | 任何有必要调查的事情。[与 IEEE1008 一致] |
| incident logging | 事件日志 | 记录所发生的（如在测试过程中）事件的详细情况 |
| incident management | 事件管理 | 识别、调查、采取行动和处理事件的过程。该过程包含对事件进行记录、分类并辨识其带来的影响。[IEEE 1044] |
| incident management tool | 事件管理工具 | 辅助记录事件并对事件进行状态跟踪的工具。这种工具常常具有面向工作流的特性，以跟踪和控制事件的资源分配、更正和再测试， 并提供报表。参见 defect management tool |
| incident report | 事件报告 | 报告任何需要调查的事件（如在测试过程中需要调查的事件）的文档。[IEEE829] |
| incremental development model | 增量开发模型 | 一种开发生命周期。项目被划分为一系列增量，每一增量都交付整个项目需求中的一部分功能。需求按优先级进行划分，并按优先级在适当的增量中交付。在这种生命周期模型的一些版本中（但不是全部），每个子项目均遵循一个“微型的 V 模型”，具有自有的设计、编码和测试阶段 |
| incremental testing | 增量测试 | 每次集成并测试一个或若干组件/系统，直到所有组件/系统都已经被集成或测试的一种测试 |
| independence | 独立 | 职责分离，有助于客观地进行测试。 [DO-178b] |
| infeasible path | 不可达路径 | 通过任何输入都无法执行到的路径 |
| informal review | 非正式评审 | 一种不基于正式（文档化）过程的评审 |
| input | 输入 | 被组件读取的变量（无论存储于组件之内还是之外） |
| input domain | 输入域 | 有效输入的集合。参见 domain |
| input value | 输入值 | 输入的一个实例。参见 input |
| inspection | 审查 | 一种同级评审，通过检查文档以检测缺陷，例如不符合开发标准，不符合更上层的文档等。这是最正式的评审技术，因此总是基于文档化的过程。[IEEE610，IEEE1028] 参见 peer review |
| inspection leader | 审查负责人 | 参见 moderator |
| inspector | 检视人/审查员 | 参见 reviewer |
| installability | 可安装性 | 软件产品在指定环境下进行安装的性能。[ISO9126]参见 portability |
| installability testing | 可安装性测试 | 测试软件产品可安装性的过程。参见 portability testing |

续表

| | | |
|---|---|---|
| I | | |
| installation guide | 安装指南 | 帮助安装人员完成安装过程的使用说明，可存放在任何合适的介质上。可能是操作指南、详细步骤、安装向导或任何其他类似的过程描述 |
| installation wizard | 安装向导 | 帮助安装人员完成安装过程的软件，可存放在任何合适的介质上。它通常会运行安装过程、反馈安装结果，并提示安装选项 |
| instrumentation | 探测 | 在程序中插入附加代码，以便在程序执行时收集其执行信息。例如，用于度量代码覆盖 |
| instrumenter | 探测工具 | 用于执行探测的软件工具 |
| intake test | 预测试 | 冒烟测试的一种特例，用于决定组件/系统是否能够进行更深入的测试。通常在测试执行的初始阶段实施 |
| integration | 集成 | 把组件/系统合并为更大部件的过程 |
| integration testing | 集成测试 | 一种旨在暴露接口以及集成组件/系统间交互时存在的缺陷的测试。参见 component integration testing, system integration testing |
| integration testing in the large | 系统集成测试 | 参见 system integration testing |
| integration testing in the small | 组件集成测试 | 参见 component integration testing |
| interface testing | 接口测试 | 一种集成测试类型，注重于测试组件/系统之间的接口 |
| interoperability | 互操作性 | 软件产品与一个或多个指定组件/系统进行交互的能力。[ISO9126] 参见 functionality |
| interoperability testing | 互操作性测试 | 判定软件产品可交互性的测试过程。参见 functionality testing |
| invalid testing | 无效性测试 | 使用应该被组件/系统拒绝的输入值进行的测试。参见 error tolerance |
| isolation testing | 隔离测试 | 将组件与其周边组件隔离后进行的测试。如果有必要，使用桩（stubs）或驱动器（drivers）来模拟周边程序 |
| item transmittal report | 版本发布报告 | 参见 release note |
| iterative development model | 迭代开发模型 | 一种开发生命周期，项目被划分为大量迭代过程。一次迭代是一个完整的开发循环，并（对内或对外）发布一个可执行的产品，这是正在开发的最终产品的一个子集，通过不断迭代最终成型的产品 |
| K | | |
| key performance indicator | 关键性能指标 | 参见 performance indicator |

续表

| | | |
|---|---|---|
| **K** | | |
| keyword driven testing | 关键字驱动测试 | 一种脚本编写技术，所使用的数据文件不单包含测试数据和预期结果，还包含与被测程序相关的关键词。用于测试的控制脚本通过调用特别的辅助脚本来解释这些关键词 |
| **L** | | |
| LCSAJ | LCSAJ | linear code sequence and jump，线性代码序列和跳转。包含以下 3 项（通常通过源代码清单的行号来识别）：可执行语句的线性序列的开始、结束以及在线性序列结尾控制流所转移到的目标行 |
| LCSAJ coverage | LCSAJ 覆盖 | 测试套件所检测的组件的 LCSAJ 百分比。LCSAJ 达到 100%意味着决策覆盖（decision coverage）为 100% |
| LCSAJ testing | LCSAJ 测试 | 一种白盒测试设计技术，其测试用例用于执行 LCSAJ |
| learnability | 易学性 | 软件产品具有的易于用户学习的能力。[ISO9126] 参见 usability |
| level test plan | 级别测试计划 | 通常用于一个测试级别（test level）的测试计划。参见 test plan |
| link testing | 组件集成测试 | 参见 component integration testing |
| load testing | 负载测试 | 一种通过增加负载来测量组件或系统的测试方法。例如，通过增加并发用户数和（或）事务数量来测量组件或系统能够承受的负载。参见 stress testing |
| logic-coverage testing | 逻辑覆盖测试 | 参见 white box testing [Myers] |
| logic-driven testing | 逻辑驱动测试 | 参见 white box testing |
| logical test case | 逻辑测试用例/抽象测试用例 | 参见 high level test case |
| low level test case | 详细测试用例 | 具有具体的（实现级别 implementation level）输入数据和预期结果的测试用例。抽象测试用例中所使用的逻辑运算符被替换为对应于逻辑运算符作用的实际值。参见 high level test case |
| **M** | | |
| maintenance | 维护 | 软件产品交付后对其进行的修改，以修正缺陷，改善性能或其他属性，或者使其适应新的环境。[IEEE1219] |
| maintenance testing | 维护测试 | 针对运行系统的更改，或者新的环境对运行系统的影响而进行的测试 |
| maintainability | 可维护性 | 软件产品是否易于更改，以便修正缺陷，满足新的需求，使以后的维护更简单或者适应新的环境。[ISO 9126] |

续表

| M | | |
|---|---|---|
| maintainability testing | 可维护性测试 | 判定软件产品的可维护性的测试过程 |
| management review | 管理评审 | 由管理层或其代表执行的对软件采购、供应、开发、运作或维护过程的系统化评估，包括监控过程、判断计划和进度表的状态、确定需求及其系统资源分配，或评估管理方式的效用，以达到正常运作的目的。[IEEE610, IEEE1028] |
| master test plan | 主测试计划 | 通常针对多个测试级别的测试计划。参见 testplan |
| maturity | 成熟度 | （1）组织在其过程和工作实践上的有效性和高效性的能力。 参见 Capability Maturity Model, Test Maturity Model。（2）软件产品在存在缺陷的情况下避免失效的能力。 [ISO9126] 参见 reliability |
| measure | 测量 | 测度时赋予实体某个属性的数值或类别。[ISO14598] |
| measurement | 测度 | 给实体赋予一个数值或类别以描述其某个属性的过程。 [ISO14598] |
| measurement scale | 度量标准 | 约束数据分析类型的标准 |
| memory leak | 内存泄露 | 程序的动态存储分配逻辑存在的缺陷，导致内存使用完毕后不能收回而不可用，最终导致程序因为内存缺乏而运行失败（fail） |
| metric | 度量 | 测量所使用的方法或者度量标准（measurement scale）。[ISO14598] |
| migration testing | 移植测试 | 参见 conversion testing |
| milestone | 里程碑 | 项目过程中预定义的（中间的）交付物和结果就绪的时间点 |
| mistake | 错误 | 参见 error |
| moderator | 主持人 | 负责检视或其他评审过程的负责人或主要人员 |
| modified condition decision coverage | 改进的条件判定覆盖 | 参见 condition determination coverage |
| modified condition decision testing | 改进的条件判定测试 | 参见 condition determination coverage testing |
| modified multiple condition coverage | 改进的复合条件覆盖 | 参见 condition determination coverage |
| modified multiple condition testing | 改进的复合条件测试 | 参见 condition determination coverage testing |
| module | 模块 | 参见 component |
| module testing | 模块测试 | 参见 component testing |
| monitor | 监测器/监视器 | 与被测组件/系统同时运行的软件工具或硬件设备，对组件/系统的行为进行监视、记录和分析。[IEEE610] |
| monitoring tool | 监测工具/监视工具 | 参见 monitor |

续表

| | | |
|---|---|---|
| **M** | | |
| multiple condition | 复合条件/多重条件 | 参见 compound condition |
| multiple condition coverage | 复合条件覆盖 | 测试套覆盖的一条语句内的所有单条件结果组合的百分比。100%复合条件覆盖意味着 100%条件判定覆盖（condition determination coverage） |
| multiple condition testing | 复合条件测试 | 一种白盒测试设计技术，测试用例用来覆盖一条语句中的单条件所有可能的结果组合 |
| mutation analysis | 变异分析 | 一种确定测试套件完整性的方法，即判定测试套件能够区分程序与其微变体之间区别的程度 |
| mutation testing | 变异测试 | 参见 back-to-back testing |
| **N** | | |
| N-switch coverage | N 切换覆盖 | N+1 个转换的序列在一个测试套件中被覆盖的百分比。[Chow] |
| N-switch testing | N 切换测试 | 一种状态转换测试的形式，其测试用例执行 N+1 个转换的所有有效序列。 [Chow] 参见 state transition testing |
| negative testing | 逆向测试 | 一种旨在表现组件/系统不能正常工作的测试。逆向测试取决于测试人员的想法、态度，而与特定的测试途径或测试设计技术无关，例如使用无效输入值测试或在异常情况下进行测试。 [Beizer] |
| non-conformity | 不一致 | 没有实现指定的需求。 [ISO9000] |
| non-functional requirement | 非功能需求 | 与功能性无关，但与可靠性（reliability）、高效性（efficiency）、可用性（usability）、可维护性（maintainability）和可移植性（portability）等属性相关的需求 |
| non-functional testing | 非功能测试 | 对组件/系统中与功能性无关的属性（如可靠性、高效性、可用性、可维护性和可移植性）进行的测试 |
| non-functional test design techniques | 非功能测试设计技术 | 推导或选择非功能测试所需测试用例的过程，此过程依据对组件/系统的规格说明进行分析，而不考虑其内部结构。参见 black box test design technique |
| **O** | | |
| off-the-shore software | 现货软件 | 面向大众市场（即大量用户）开发的软件产品，并且以相同的形式交付给许多客户 |
| operability | 可操作性 | 软件产品被用户操作或控制的能力。 [ISO9126] 参见 usability |
| operational environment | 运行环境 | 用户或客户现场所安装的硬件和软件产品，被测组件/系统将在此环境下使用。软件可能包括操作系统、数据库管理系统和其他应用程序 |
| operational profile testing | 运行概况测试 | 对系统运作模型（执行短周期任务）及其典型应用概率的统计测试。[Musa] |

续表

| | | |
|---|---|---|
| O | | |
| operational testing | 运行测试 | 在组件/系统的运作环境下对其进行评估的一种测试。[IEEE 610] |
| oracle | 基准 | 参见 test oracle |
| outcome | 结果 | 参见 result |
| output | 输出 | 组件填写的一个变量（无论存储在组件内部还是外部） |
| output domain | 输出域 | 可从中选取有效输出值的集合。参见 domain |
| output value | 输出值 | 输出的一个实例/实值。参见 output |
| P | | |
| pair programming | 结对编程 | 一种软件开发方式，组件的代码（开发和/或测试）由两名程序员在同一台计算机上共同编写。这意味着实时地执行代码评审 |
| pair testing | 结对测试 | 两个人员，比如两个测试人员、一个开发人员和一个测试人员或一个最终用户和一个测试人员，一起寻找缺陷。一般地，他们使用同一台计算机并在测试期间交替操控 |
| partition testing | 划分测试 | 参见 equivalence partitioning [Beizer] |
| pass | 通过 | 如果一个测试的实际结果与预期结果相符，则认为此测试通过 |
| pass/fail criteria | 通过/失败准则 | 用于判定测试项（功能）或特性通过或失败的决策规则。[IEEE829] |
| path | 路径 | 组件/系统从入口（entry point）到出口（exit point）的一系列事件（如可执行语句） |
| path coverage | 路径覆盖 | 测试套件执行的路径所占的百分比。100%的路径覆盖意味着 100%的线性代码序列和跳转（LCSAJ）覆盖 |
| path sensitizing | 路径感知 | 选择一组输入值，以强制执行某指定路径 |
| path testing | 路径测试 | 一种白盒测试设计技术，设计的测试用例用于执行路径 |
| peer review | 同行评审 | 由研发产品的同事对软件产品进行的评审，目的在于识别缺陷并改进产品。如审查（inspetion）、技术评审（technical review）和走查（walkthrough） |
| performance | 性能 | 组件/系统在给定的处理周期和吞吐率(throughput rate)等约束下，完成指定功能的程度。[IEEE610] 参见 efficiency |
| performance indicator | 性能指标 | 一种有效性（effectiveness）和/或高效性（efficiency）的高级（抽象）度量单位，用于指导和控制开发进展。如软件交付时间的偏差（lead-time slip for software devlopment）。[CMMI] |
| performance testing | 性能测试 | 判定软件产品性能的测试过程。参见 efficiency testing |

续表

| P | | |
|---|---|---|
| performance testing tool | 性能测试工具 | 一种支持性能测试的工具，通常有两个功能：负载生成（load gerneartion）和测试事务（test transation）测量。负载生成可以模拟多用户或者大量输入数据。执行时，对选定的事务的响应时间进行测量并被记录。性能测试工具通常会生成基于测试日志的报告以及负载-响应时间图表 |
| phase test plan | 阶段测试计划 | 通常用于一个测试阶段的测试计划。参见 testplan |
| portability | 可移植性 | 软件产品在不同硬件或软件环境之间迁移的简易性。[ISO9126] |
| portability testing | 可移植性测试 | 判定软件产品可移植性的测试过程 |
| postcondition | 后置条件 | 执行测试或测试步骤后必须满足的环境和状态条件 |
| post-execution comparison | 执行后比较 | 实际值与预期值的比较，在软件运行结束后执行 |
| precondition | 前置条件 | 对组件/系统执行特定测试或测试步骤之前所必须满足的环境和状态条件 |
| predicted outcome | 预期结果 | 参见 expected result |
| pretest | 预测试 | 参见 intake test |
| priority | 优先级 | 赋予某项（业务）重要性的级别，如缺陷 |
| probe effect | 探测影响 | 在测试时由于测试工具（如性能测试工具或监测器）对组件/系统产生的影响。如使用性能测试工具可能会使系统的性能有小幅度降低 |
| problem | 问题 | 参见 defect |
| problem management | 问题管理 | 参见 defect management |
| problem report | 问题报告 | 参见 defect report |
| process | 过程 | 一组将输入转变为输出的相关活动。[ISO12207] |
| process cycle test | 过程周期测试 | 一种黑盒测试设计技术，设计的测试用例用于执行业务流程或过程。[Tmap] |
| product risk | 产品风险 | 与测试对象有直接关系的风险。参见 risk |
| project | 项目 | 一个项目是一组以符合特定需求为目的的，相互协同的，具有开始和结束时间的受控活动。这些特定需求包括限定的周期、成本和资源。[ISO9000] |
| project risk | 项目风险 | 与（测试）项目的管理与控制相关的风险。参见 risk |
| program instrumenter | 程序插装器 | 参见 instrumenter |
| program testing | 程序测试 | 参见 component testing |
| project test plan | 项目测试计划 | 参见 master test plan |
| pseudo-random | 伪随机 | 一个表面上随机的序列，但事实上是根据预定的序列生成的 |

续表

| | | |
|---|---|---|
| Q | | |
| quality | 质量 | 组件、系统或过程满足指定需求或用户/客户需要及期望的程度。 [IEEE610] |
| quality assurance | 质量保证 | 质量管理的组成部分，提供达到质量要求的可信程度。[ISO9000] |
| quality attribute | 质量属性 | 影响某项质量的特性或特征。 [IEEE610] |
| quality characteristic | 质量特征 | 参见质量属性（quality attribute） |
| quality management | 质量管理 | 在质量方面指导和控制一个组织的协同活动。通常包括建立质量策略和质量目标、质量计划、质量控制、质量保证和质量改进。 [ISO9000] |
| R | | |
| random testing | 随机测试 | 一种黑盒测试设计技术，选择测试用例以匹配某种运行概貌情况（可能使用伪随机生成算法）。这种技术可用于测试非功能性的属性，如可靠性和性能 |
| recorder | 记录员 | 参见 scribe |
| record/playback tool | 录制/回放工具 | 参见 capture/playback tool |
| recoverability | 可恢复性 | 软件产品失效（failure）后，重建其特定性能级别以及恢复数据的能力。 [ISO9126] 参见 reliability |
| recoverability testing | 可恢复性测试 | 判定软件产品可恢复性的测试过程。参见 reliability testing |
| recovery testing | 恢复测试 | 参见 recoverability testing |
| regression testing | 回归测试 | 测试先前测试过并修改过的程序，确保更改没有给软件其他未改变的部分带来新的缺陷（defect）。软件修改后或使用环境变更后要执行回归测试 |
| regulation testing | 规范性测试 | 参见 compliance testing |
| release note | 发布说明 | 标识测试项、测试项配置、目前状态及其他交付信息的文档，这些交付信息是由开发、测试和可能的其他风险承担者在测试执行阶段开始的时候提交的。[ISO 9126] |
| reliability | 可靠性 | 软件产品在一定条件下（规定的时间或操作次数等），执行其必需的功能的能力。 [ISO9126] |
| reliability testing | 可靠性测试 | 判定软件产品可靠性的测试过程 |
| replaceability | 可替换性 | 在相同环境下，软件产品取代另一指定软件产品以达到相同目的的能力。[ISO9126] 参见 portability |
| requirement | 需求 | 系统必须满足的，为用户解决问题或达到目的之条件或者能力。通过系统或者系统的组件的运行以满足合同、标准、规范或其他指定的正式文档定义的要求。[IEEE610] |

续表

| R | | |
|---|---|---|
| requirements-based testing | 基于需求的测试 | 根据需求推导测试目标和测试条件以设计测试用例的方法。例如，执行特定功能的测试或探测诸如可靠性和可用性等非功能性属性的测试 |
| requirements management tool | 需求管理工具 | 一种支持需求记录、需求属性（如优先级）和注解的工具，能够通过多层次需求和需求变更管理达到可追踪性。一些需求管理工具还支持静态分析，如一致性检查以及预定义的需求规则之间的冲突 |
| requirements phase | 需求阶段 | 在软件生命周期中定义和文档化软件产品需求的阶段。 [IEEE610] |
| resource utilization | 资源使用 | 软件产品在规定的条件下执行其功能时，使用适当数量和类型资源的能力。例如，程序使用的主存储器和二级存储器容量，需要的临时或溢出文件的大小。[ISO 9126] 参见 efficiency |
| resource utilization testing | 资源使用测试 | 判定软件产品资源使用的测试过程。参见 efficiency testing |
| result | 结果 | 测试执行的成果，包括屏幕输出、数据更改、报告和发出的通信消息。参见 actual result, expected result |
| resumption criteria | 继续准则 | 在重新启动被中断（或者延迟）的测试时，必须重复执行的测试活动。 [After IEEE 829] |
| re-testing | 再测试 | 重新执行上次失败的测试用例，以验证纠错的正确性 |
| review | 评审 | 对产品或产品状态进行的评估，以确定与计划的结果所存在的误差，并提供改进建议。例如，管理评审（management review）、非正式评审（informal review）、技术评审（technical review）、审查（inspection）和走查（walkthrough）。 [After IEEE1028] |
| reviewer | 评审人 | 参与评审的人员，辨识并描述被评审产品或项目中的异常。在评审过程中，可以选择评审人员从不同角度评审或担当不同角色 |
| review tool | 评审工具 | 对评审过程提供支持的工具。典型的功能包括计划评审、跟踪管理、通信支持、协同评审以及对具体度量（单位）收集与报告的存储库 |
| risk | 风险 | 将会导致负面结果的因素。通常表达成可能的（负面）影响 |
| risk analysis | 风险分析 | 评估识别出的风险以估计其影响和发生的可能性的过程 |
| risk-based testing | 基于风险的测试 | 倾向于探索和提供有关产品风险信息的测试。[After Gerrard] |
| risk control | 风险控制 | 为降低风险到或控制风险在指定级别而达成的决议和实施防范（度量）措施的过程 |

续表

| | | |
|---|---|---|
| R | | |
| risk identification | 风险识别 | 使用技术手段[如头脑风暴（brainstorming）、检验表（checklists）和失败历史记录（failurehistory）]标识风险的过程 |
| risk management | 风险管理 | 对风险进行标识、分析、优先级划分和控制所应用的系统化过程和实践 |
| risk mitigation | 风险缓解 | 参见 risk control |
| robustness | 健壮性 | 在出现无效输入或压力环境条件下，组件/系统能够正常工作的程度。[IEEE610] 参见 error-tolerance, fault-tolerance |
| robustness testing | 健壮性测试 | 判定软件产品健壮性的测试 |
| root cause | 根本原因 | 导致不一致的根本因素，并具有通过过程改进彻底清除的可能 |
| S | | |
| safety | 安全性 | 软件产品在特定的使用环境中， 达到对人、业务、软件、财产或环境可接受的危害风险级别的能力[ISO 9126 ] |
| safety testing | 安全性测试 | 判定软件产品安全性的测试 |
| sanity test | 健全测试 | 参见冒烟测试（smoke test） |
| scalability | 可扩展性 | 软件产品可被升级以容纳更多负载的能力[Gerrard] |
| scalability testing | 可扩展性测试 | 判定软件产品可扩展性的测试 |
| scenario testing | 场景测试 | 参见用例测试（use case testing） |
| scribe | 记录员 | 在评审会议中将每个提及的缺陷和任何过程改进建议记录到日志表单上的人员，记录员要确保日志表单易于阅读和理解 |
| scripting language | 脚本语言 | 一种用于编写可执行测试脚本（这些脚本被测试执行工具使用，如录制/回放工具）的编程语言 |
| security | 安全性 | 软件产品防止对程序和数据未授权访问（无论是有意的还是无意的） 的能力的属性。 [ISO9126]参见功能性（functionality） |
| security testing | 安全性测试 | 判定软件产品安全性的测试，参见功能性测试（functionality testing） |
| security testing tool | 安全性测试工具 | 测试安全特性和脆弱性的工具 |
| security tool | 安全性工具 | 提高运行安全性的工具 |
| serviceability testing | 服务能力测试 | 参见维护能力测试（maintainability test） |
| severity | 严重性 | 缺陷对组件/系统的开发或运行造成的影响程度。[IEEE 610] |
| simulation | 模拟 | 一个实际或抽象系统的特定行为特征由另一个系统来代表。 [ISO2382/1] |

续表

| S | | |
|---|---|---|
| simulator | 模拟器 | 测试时所使用的设备、计算机程序或者系统，当提供一套控制的输入集时它们的行为或运行与给定的系统相似。 [IEEE610 DO178b]。参见模拟器（emulator） |
| site acceptance testing | 现场验收测试 | 用户/客户在他们现场进行的验收测试，以判定组件/系统是否符合他们的需求和业务流程，通常包括软件和硬件 |
| smoke test | 冒烟测试 | 所有定义的/计划的测试用例的一个子集，它覆盖组件/系统的主要功能，以确保程序的绝大部分关键功能正常工作，但忽略细节部分。每日构建和冒烟测试是业界的最佳实践。参见预测试（intake test） |
| software | 软件 | 计算机程序、过程和可能与计算机系统运行相关的文档和数据 |
| software feature | 软件特性 | 参见特性（feature） |
| software quality | 软件质量 | 软件产品的功能和特性总和， 能够达到规定的或隐含的需求。 [ISO9126] |
| software quality characteristic | 软件质量特性 | 参见质量属性（quality attribute） |
| software test incident | 软件测试事件 | 参见事件（incident） |
| software test incident report | 软件测试事件报告 | 参见事件报告（incident report） |
| software usability measurement inventory（SUMI） | 软件可用性度量调查表 | 一种基于调查表的可用性测试技术，以评估组件/系统的可用性，如用户满意度。[Veenendaal] |
| source statement | 源语句 | 参见语句（statement） |
| specification | 规格说明 | 说明组件/系统的需求、设计、行为或其他特征的文档，常常还包括判断是否满足这些条款的方法。理想情况下，文档是以全面、精确、可验证的方式进行说明的 |
| specification-based testing | 基于规格说明的测试 | 参见黑盒测试（black-box testing） |
| specification-based test design technique | 基于规格说明的测试设计技术 | 参见黑盒测试设计技术（black-box test design technique） |
| specified input | 特定的输入 | 在规格说明中预测结果的输入 |
| stability | 稳定性 | 软件产品避免因更改后导致非预期结果的能力。[ISO9126]参见可维护性（maintainability） |

续表

| S | | |
|---|---|---|
| standard software | 标准软件 | 参见现货软件（off-the-shelf software） |
| standards testing | 标准测试 | 参见一致性测试（compliance testing） |
| state diagram | 状态图 | 一种图表， 描绘组件/系统所能呈现的状态，并显示导致或产生从一个状态转变到另一个状态的事件或环境 |
| state table | 状态表 | 一种表格，显示每个状态的有效和无效的转换及可能的伴随事件 |
| state transition | 状态转换 | 组件/系统的两个状态之间的转换 |
| state transition testing | 状态转换测试 | 一种黑盒测试设计技术，所设计的测试用例用来执行有效和无效的状态转换。参见 N 切换测试（N-switch testing） |
| statement | 语句 | 编程语言的一个实体，一般是最小的、不可分割的执行单元 |
| statement coverage | 语句覆盖 | 由测试套件运行的可执行语句的百分比 |
| statement testing | 语句测试 | 一种白盒测试设计技术，所设计的测试用例用来执行语句 |
| static analysis | 静态分析 | 分析软件工件（如需求或代码），而不执行这些工作产品 |
| static analysis tool | 静态分析工具 | 参见静态分析器（static analyzer） |
| static analyzer | 静态分析器 | 执行静态分析的工具 |
| static code analysis | 静态代码分析 | 分析软件的源代码而不执行软件 |
| static code analyzer | 静态代码分析器 | 执行静态代码分析的工具。工具对源代码的一些特性进行检查，如对编码规范的遵循、质量度量或数据流异常等 |
| static testing | 静态测试 | 对组件/系统进行规格或实现级别的测试，而不是执行这个软件，如代码评审或静态代码分析 |
| statistical testing | 统计测试 | 用输入的统计分布模型来构造有代表性的测试用例的一种测试设计技术。参见运行概貌测试（operational profile testing） |
| status accounting | 状态记录 | 配置管理的一个要素，包括记录和报告有效地管理配置所需的信息。这些信息包括被认可的配置标识的列表、提议的配置变更的状态和被认可的变更的实施状态。[IEEE610] |
| storage | 存储 | 参见资源利用（resource utilization） |
| storage testing | 存储测试 | 参见资源利用测试（resource utilization testing） |
| stress testing | 压力测试 | 在规定的或超过规定的需求条件下测试组件/系统，以对其进行评估。[IEEE610] 参见（load testing） |
| structure-based techniques | 基于结构的技术 | 参见白盒测试设计技术（white box test design technique） |

续表

| | | |
|---|---|---|
| S | | |
| structural coverage | 结构覆盖 | 基于组件/系统内部结构的覆盖度量 |
| structural test design technique | 结构测试设计技术 | 参见白盒测试设计技术（white-box test design technique） |
| structural testing | 结构测试 | 参见白盒测试（white-box testing） |
| structured walkthrough | 结构走查 | 参见走查（walkthrough） |
| stub | 桩 | 一个软件组件框架的实现或特殊目的的实现，用于开发和测试另一个调用或依赖于该组件的组件。它代替了被调用的组件。 [IEEE610] |
| subpath | 子路径 | 组件中的可执行语句序列 |
| suitability | 适用性 | 软件产品为特定任务和用户目标提供一套合适功能的能力。 [ISO9126]参见功能性（functionality） |
| suspension criteria | 暂停准则 | 用来（暂时性地）停止对测试条目进行的所有或部分测试活动的准则。[IEEE829] |
| syntax testing | 语法测试 | 一种黑盒测试设计技术，测试用例的设计是以输入域和（或）输出域的定义的依据 |
| system | 系统 | 组织在一起实现一个特定功能或一组功能的一套组件。[IEEE610] |
| system integration testing | 系统集成测试 | 测试系统和包的集成；测试与外部组织（如电子数据交换、国际互联网）的接口 |
| system testing | 系统测试 | 测试集成系统以验证它是否满足指定需求的过程。[Hetzel] |
| T | | |
| technical review | 技术评审 | 一种同行间的小组讨论活动，主要为了对所采用的技术实现方法达成共识。[ Gilb 和 Graham, IEEE 1028 ] 参见同行评审（peer review） |
| test | 测试 | 一个或更多测试用例的集合 [IEEE829] |
| test approach | 测试方法 | 针对特定项目的测试策略的实现，通常包括根据测试项目的目标和风险进行评估之后所做的决策、测试过程的起点、采用的测试设计技术、退出准则和所执行的测试类型 |
| test automation | 测试自动化 | 应用软件来执行或支持测试活动，如测试管理、测试设计、测试执行和结果检验 |
| test basis | 测试依据 | 能够从中推断出组件/系统需求的所有文档。测试用例是基于这些文档的。只能通过正式的修正过程来修正的文档称为固定测试依据。 [TMap] |
| test bed | 测试台 | 参见测试环境（test environment） |
| test case | 测试用例 | 为特定目标或测试条件（例如，执行特定的程序路径，或是验证与特定需求的一致性）而制定的一组输入值、执行入口条件、预期结果和执行出口条件。[IEEE610] |

续表

| T | | |
|---|---|---|
| test case design technique | 测试用例设计技术 | 参见测试设计技术（test design technique） |
| test case specification | 测试用例规格说明 | 为测试项指定一套测试用例（目标、输入、测试动作、期望结果、执行预置条件）的文档。[IEEE829] |
| test case suite | 测试用例集 | 参见测试套（test suite） |
| test charter | 测试章程 | 对测试目标的陈述，还可能包括关于如何进行测试的测试思路。测试章程通常用在探索测试中。参见探索测试（exploratory testing） |
| test closure | 测试结束 | 从已完成的测试活动中收集数据，总结基于测试件及相关事实和数据的测试结束阶段，包括对测试件的最终处理和归档，以及测试过程评估（包含测试评估报告的准备）。参见测试过程（test process） |
| test comparator | 测试比较器 | 执行自动测试比较的测试工具 |
| test comparison | 测试对比 | 区分被测组件/系统产生的实际结果和期望结果的差异的过程。测试对比可以在测试执行时进行（动态比较），或在测试执行之后进行 |
| test completion criteria | 测试完成准则 | 参见退出准则（exit criteria） |
| test condition | 测试条件 | 组件/系统中能被一个或多个测试用例验证的条目或事件。如功能、事务、特性、质量属性或者结构化元素 |
| test control | 测试控制 | 当监测到与预期情况背离时，制定和应用一组修正动作以使测试项目保持正常进行的测试管理工作。参见测试管理（test management） |
| test coverage | 测试覆盖 | 参见覆盖（coverage） |
| test cycle | 测试周期 | 针对一个可分辨的测试对象发布版本而执行的测试过程 |
| test data | 测试数据 | 在测试执行之前存在的数据（如在数据库中），这些数据与被测组件/系统相互影响 |
| test data preparation tool | 测试数据准备工具 | 一种测试工具，用于从已存在的数据库中挑选数据，或创建、生成、操作和编辑数据以备测试 |
| test design | 测试设计 | 参见测试设计规格说明（test design specification） |
| test design specification | 测试设计规格说明 | 为一个测试条目指定测试条件（覆盖项）、具体测试方法并识别相关高层测试用例的文档 |
| test design technique | 测试设计技术 | 用来衍生和/或选择测试用例的步骤 |
| test design tool | 测试设计工具 | 通过生成测试输入来支持测试设计的工具。 测试输入可能来源于 CASE 工具库（如需求管理工具）中包含的规格，工具本身包含的特定测试条件 |

续表

| T | | |
|---|---|---|
| test driver | 测试驱动器 | 参见驱动器（driver） |
| test driven development | 测试驱动开发 | 在开发软件之后，运行测试用例之前，首先开发并自动化这些测试用例的一种软件开发方法 |
| test environment | 测试环境 | 执行测试需要的环境，包括硬件、仪器、模拟器、软件工具和其他支持要素 |
| test evaluation report | 测试评估报告 | 在测试过程的结尾用来总结所有的测试活动和结果的文档。也包括测试过程的评估和吸取的教训 |
| test execution | 测试执行 | 对被测组件/系统执行测试，产生实际结果的过程 |
| test execution automation | 测试执行自动化 | 使用软件（如捕捉/回放工具）来控制测试的执行、实际结果和期望结果的对比、测试预置条件的设置和其他的测试控制和报告功能 |
| test execution phase | 测试执行阶段 | 软件开发生命周期的一个阶段，在这个阶段里执行软件产品的组件，并评估软件产品以确定是否满足需求 |
| test execution schedule | 测试执行时间表 | 测试过程的执行计划。这些测试过程包含在测试执行时间表中，执行时间表列出了执行任务间的关联和执行的顺序 |
| test execution technique | 测试执行技术 | 用来执行实际测试的方法，包括手工的和自动的 |
| test execution tool | 测试执行工具 | 使用自动化测试脚本执行其他软件（如捕捉/回放）的一种测试工具。[Fewster 和 Graham] |
| test fail | 测试失败 | 参见失败（fail） |
| test generator | 测试产生器 | 参见测试数据准备工具（test data preparation tool） |
| test harness | 测试用具 | 包含执行测试需要的桩和驱动的测试环境 |
| test incident | 测试事件 | 参见事件（incident） |
| test incident report | 测试事件报告 | 参见事件报告（incident report） |
| test infrastructure | 测试基础设施 | 执行测试所需的组成物件，包括测试环境、测试工具、办公环境和过程 |
| test input | 测试输入 | 在测试执行过程中，测试对象从外部源接收到的数据。外部源可以是硬件、软件或人 |
| test item | 测试项 | 需被测试的单个要素。通常是一个测试对象包含多个测试项。参见测试对象（test object） |
| test item transmittal report | 测试项移交报告 | 参见发布说明（release note） |
| test leader | 测试组长 | 参见测试经理（test manager） |
| test level | 测试级别 | 统一组织和管理的一组测试活动。测试级别与项目的职责相关联。例如，测试级别有组件测试、集成测试、系统测试和验收测试。[TMap] |
| test log | 测试日志 | 按时间顺序排列的有关测试执行所有相关细节的记录 |

续表

| T | | |
|---|---|---|
| test logging | 测试记录 | 把测试执行信息写进日志的过程 |
| test manager | 测试经理 | 负责测试和评估测试对象的人。他（她）指导、控制、管理测试计划及调整对测试对象的评估 |
| test management | 测试管理 | 计划、估计、监控和控制测试活动，通常由测试经理来执行 |
| test management tool | 测试管理工具 | 对测试过程中的测试管理和控制部分提供支持的工具。它通常有如下功能：测试件的管理、测试计划的制订、结果记录、过程跟踪、事件管理和测试报告 |
| test maturity model （TMM） | 测试成熟度模型 | 测试过程改进的五级阶段框架，它与能力成熟度模型（CMM）相关，后者描述了有效测试过程的关键要素 |
| test monitoring | 测试监控 | 处理与定时检查测试项目状态等活动相关的测试管理工作。准备测试报告来比较实际结果和期望结果。参见测试管理（test management） |
| test object | 测试对象 | 需要测试的组件或系统。参见测试项（test item） |
| test objective | 测试目标 | 设计和执行测试的原因或目的 |
| test oracle | 测试准则 | 在测试时确定预期结果与实际结果进行比较的源。一个准则可能是现有的系统（用作基准），一份用户手册，或者是个人的专业知识，但不可以是代码。[Adrion] |
| test outcome | 测试结果 | 参见结果（result） |
| test pass | 测试通过 | 参见通过（pass） |
| test performance indicator | 测试绩效指标 | 一种高级别的度量，表明需要满足的某种程度的目标值或准则。通常与过程改进的目标相关。例如，缺陷探测率 |
| test phase | 测试阶段 | 组成项目的一个可管理阶段的一组独特的测试活动。例如，某测试级别的执行活动。[Gerrard] |
| test plan | 测试计划 | 描述预期测试活动的范围、方法、资源和进度的文档。它标识了测试项、需测试的特性、测试任务、任务负责人、测试人员的独立程度、测试环境、测试设计技术、测试的进入和退出准则和选择的合理性、需要紧急预案的风险，是测试策划过程的一份记录。[IEEE 829] |
| test planning | 测试策划 | 制订或更新测试计划的活动 |
| test policy | 测试方针 | 描述有关组织测试的原则、方法和主要目标的高级文档 |
| test point analysis （TPA） | 测试点分析（TPA） | 基于功能点分析的一种公式化测试估计方法。[TMap] |
| test procedure | 测试规程 | 参见测试规程规范（test procedure specification） |
| test procedure specification | 测试规程规格说明 | 规定了执行测试的一系列行为的文档。也称为测试脚本或手工测试脚本。[IEEE 829] |

续表

| T | | |
|---|---|---|
| test process | 测试过程 | 基本的测试过程包括计划、规约、执行、记录、检查完全性和测试结束活动 |
| test process improvement（TPI） | 测试过程改进（TPI） | 用于测试过程改进的一个连续框架，描述了有效测试过程的关键要素，特别针对于系统测试和验收测试 |
| test record | 测试记录 | 参见测试日志（test log） |
| test recording | 书写测试记录 | 参见测试日志（test logging） |
| test repeatability | 测试重复性 | 一个测试的属性，表明每次执行一个测试时是否产生同样的结果 |
| test report | 测试报告 | 参见测试总结报告（test summary report） |
| test requirement | 测试需求 | 参见测试条件（test condition） |
| test run | 测试运行 | 对测试对象的特定版本执行测试 |
| test run log | 测试运行日志 | 参见测试日志（test log） |
| test result | 测试结果 | 参见结果（result） |
| test scenario | 测试场景 | 参见测试规程规约（test procedure specification） |
| test script | 测试脚本 | 通常指测试规程规约，尤其是自动化的 |
| test set | 测试集 | 参见测试套件（test suite） |
| test situation | 测试状况 | 参见测试条件（test condition） |
| test specification | 测试规约说明 | 由测试设计规约、测试用例规约和/或测试规程规约组成的文档 |
| test specification technique | 测试规约说明技术 | 参见测试设计技术（test design technique） |
| test stage | 测试阶段 | 参见测试级别（test level） |
| test strategy | 测试策略 | 一个高级文档，该文档定义了需要对程序（一个或多个项目）执行的测试级别和需要进行的测试 |
| test suite | 测试套件 | 用于被测组件/系统的一组测试用例。在这些测试用例中，一个测试的出口条件通常用作下个测试的入口条件 |
| test summary report | 测试总结报告 | 总结测试活动和结果的文档。也包括对测试项是否符合退出准则进行的评估 |
| test target | 测试目标 | 参见退出准则（exit criteria） |
| test technique | 测试技术 | 参见测试设计技术（test design technique） |
| test tool | 测试工具 | 支持一个或多个测试活动（如计划和控制、规格制定、建立初始文件和数据、测试执行和测试分析）的软件产品。[TMap ] 参见 CAST |
| test type | 测试类型 | 旨在针对特定测试目标，测试组件/系统的一组测试活动。如功能测试、易用性测试、回归测试等。一个测试类型可能发生在一个或多个测试级别或测试阶段上。 [TMap] |

续表

| | | |
|---|---|---|
| T | | |
| testability | 可测试性 | 软件产品修改后被测试的能力。[ISO9126] 参见可维护性（maintainability） |
| testability review | 可测试性评审 | 详细检查测试依据，以判定测试依据在测试过程中作为输入文档是否达到质量要求 |
| testable requirements | 可测的需求 | 对需求的一种程度说明，表示是可依据需求进行测试设计（以及后续的测试用例）和执行测试，以及判断是否满足需求。[IEEE610] |
| tester | 测试员 | 参与测试组件/系统的专业技术人员 |
| testing | 测试 | 包括了所有生命周期活动的过程，有静态的也有动态的。涉及计划、准备和对软件及其相关工作产品的评估，以发现缺陷来判定软件或软件的工作产品是否满足特定需求，证明它们是否符合目标 |
| testware | 测试件 | 在测试过程中产生的测试计划、测试设计和执行测试所需要的人工制品。例如，文档、 脚本、输入、预期结果、安装和清理步骤、文件、数据库、环境和任何在测试中使用的软件和工具。[Fewster 和 Graham] |
| thread testing | 线程测试 | 组件集成测试的一个版本，其中，组件的渐进式集成遵循需求子集的实现，与按层次的组件集成相反 |
| time behavior | 时间行为 | 参见性能（performance） |
| top-down testing | 自顶向下测试 | 集成测试的一种递增实现方式，首先测试最顶层的组件，其他组件使用桩来模拟，然后已被测试过的组件用于测试更低层的组件，直到最底层的组件被测试。参见集成测试（integration testing） |
| traceability | 可追溯性 | 识别文档和软件中相关联条目的能力。例如，需求与相关测试关联。参见水平可跟踪性（horizontal traceability），垂直可跟踪性（vertical traceability） |
| U | | |
| understandability | 可理解性 | 软件产品对于用户是否易于理解、软件是否适用、怎样应用于特定任务和应用的条件的能力 |
| unit | 单元 | 参见组件（component） |
| unit testing | 单元测试 | 参见组件测试（component testing） |
| unreachable code | 不可达代码 | 不能够到达因而不可能被执行的代码 |
| usability | 可用性 | 软件能被理解、学习、使用和在特定应用条件下吸引用户的能力。[ISO9126] |
| usability testing | 可用性测试 | 用来判定软件产品的可被理解、易学、易操作和在特定条件下吸引用户程度的测试 |
| use case | 用例 | 用户和系统进行对话过程中的一系列交互，能够产生实际的结果 |
| use case testing | 用例测试 | 一种黑盒测试设计技术，所设计的测试用例用于执行用户场景 |

续表

| | | |
|---|---|---|
| **U** | | |
| user acceptance testing | 用户验收测试 | 参见验收测试（acceptance testing） |
| user scenario testing | 用户场景测试 | 参见用例测试（use case testing） |
| user test | 用户测试 | 由真实用户参与的评估组件/系统可用性的测试 |
| **V** | | |
| V-model | V 模型 | 描述从需求定义到维护的整个软件开发生命周期活动的框架。V 模型说明了测试活动如何集成于软件开发生命周期的每个阶段 |
| validation | 确认 | 通过检查和提供客观证据来证实特定目的功能或应用已经实现。[ISO9000] |
| variable | 变量 | 计算机中的存储元素，软件程序通过其名称来引用 |
| verification | 验证 | 通过检查和提供客观证据来证实指定的需求是否已经满足。[ISO9000] |
| vertical traceability | 垂直可跟踪性 | 贯穿开发文档到组件层次的需求跟踪 |
| version control | 版本控制 | 参见配置控制（configuration control） |
| volume testing | 容量测试 | 使用大容量数据对系统进行的一种测试。参见资源利用测试（resource-utilization testing） |
| **W** | | |
| walkthrough | 走查 | 由文档作者逐步陈述文档内容，以收集信息并对内容达成共识。[Freedman 和 Weinberg, IEEE1028]。参见同行评审（peer review） |
| white-box test design technique | 白盒测试设计技术 | 通过分析组件/系统的内部结构来产生和/或选择测试用例的规程 |
| white-box testing | 白盒测试 | 通过分析组件/系统的内部结构进行的测试 |
| wide band delphi | 宽带德尔菲法 | 一种专家测试评估的方法，旨在集团队员的智慧来做精确的评估 |

# 参 考 文 献

刘德宝. 软件测试工程师培训教程. 北京：北京科海电子出版社，科学出版社，2009
任冬梅，陈汶宾，朱小梅. 软件测试技术基础. 北京：清华大学出版社，2008
柳纯录主编. 软件评测试教程. 北京：清华大学出版社，2005
（美）Ron Patton 著，周予滨，姚静等译. 软件测试. 北京：机械工业出版社，2002
蔡为东. 软件测试工程师面试指导. 北京：科学出版社，2007
教育部考试中心编. 全国计算机等级考试四级教程——软件测试工程师（2008 年版）. 北京：高等教育出版社，2007
陈能技. 软件测试技术大全——测试基础流行工具项目实战. 北京：人民邮电出版社，2008
崔启亮，胡一鸣编著. 国际化软件测试. 北京：电子工业出版社，2006